Design of a Practical Wireless Mesh Network Using Available Consumer Hardware

by

Jay Sethi
Matthew Wawrin
Ryan Supeene

Final report submitted in partial satisfaction of the
requirements for the degree of
Bachelor of Science
in
Computer Engineering
in the
Faculty of Engineering
of the
University of Manitoba

Faculty Supervisor:
Dr. Ekram Hossain

© Copyright Jay Sethi Matthew Wawrin Ryan Supeene, 2006

I. Abstract

The project demonstrates that the creation of a functional wireless mesh network (WMN) is possible using off the shelf components. Routers were configured to allow data to hop wirelessly from one router to the next before reaching its destination. This network was designed to handle common Internet tasks such as email, web browsing and file transfers. In addition, automation scripts allowed these routers to be easily deployed without any infrastructure requirements. Implementing such a network included hardware and software considerations needed to support wireless communications and allow clients to easily connect through the network to access another external network. Design of a routing scheme to handle traffic from a client, within the mesh network and to an external network was required.

Testing of the network was done to determine the affect the number of hops a packet must take to reach its destination and had on throughput and round trip time. The fairness between two clients at different location of the network was also tested. To do this, traffic generators and common user tasks such as file transfers were used, while varying the configuration of the network.

The network was successfully implemented reaching a maximum of four hops, limited by hardware resources of the project. Although Internet tasks could be done on the network, a memory leak with the open source driver for the wireless network devices, full testing results were not obtained.

This report outlines the configuration of the mesh router hardware and how the software system on these machines was implemented to have them behave as mesh routers, joining together to form a WMN. The report also contains the testing information that was able to be collected even with the malfunctioning driver.

II. Contributions

This project focuses on the design and implementation of a wireless mesh network using common consumer hardware, and the testing of the network designed with common internet user applications. Wireless mesh networks are becoming popular solutions in many applications. This project used consumer hardware combined with open source software, it was implemented with a low budget using easily obtainable parts. The project was split into roughly three different sections. These sections were separated into the setup and configuration of the computers, the design and configuration of the mesh router software, and testing. These sections were worked on by the whole group, but had individual leaders who made sure the work was completed. The break down of individual contributions is as follows:

Setup and Configuration of machines:

- Hardware installation: Group
- Operating system installation and setup: Jay/Matt
- Driver installation and setup: Matt/Jay
- Routing software installation: Jay/Matt
- Startup Automation: Matt

Design and Configuration of mesh router Software:

- Router configuration: Ryan/Jay
- Nat Translation for router: Jay
- Gateway resolution for router: Ryan
- Debugging: Group

Testing of Mesh network:

- Setup and Testing: Group
- Analysis of data: Ryan

Document Composition:

- Mesh Router Hardware: Ryan/Group
- Software Systems: Matt
- Packet Routing: Click Modular Routing: Jay
- System Integration: Matt/Group
- System Automation: Matt
- Testing Methodology and Analysis: Ryan

III. Acknowledgments

The authors of this report would like to thank our advisor, Dr. Ekram Hossain, for introducing the topic of wireless mesh networks to us, and giving us expert guidance throughout the project. Without his ideas and help this project would not have gone as far as it did.

We would like to thank the Internet Innovations Centre and Dr. Robert McLeod for lending us the five computer systems we used for this project. Without these machines, the scope of this project would have been severely limited.

We would also like to thank Mr. Mount-First Ng, Mr. Gordon Toole and Mr. Alan McKay for their help in selecting and ordering parts, as well as making hardware resources from the Tech. Shop available to us.

IV. Table of Contents

I. Abstract ii

II. Contributions iii

III. Acknowledgments iv

IV. Table of Contents v

V. List of Figures ix

VI. List of Tables x

VII. List of Definitions and Acronyms xi

Chapter 1 - Introduction 1

1.1 Overview 1

1.2 Purpose 1

1.3 Scope 1

1.4 Organization of Report 2

Chapter 2 - Background 3

2.1 Network Layer Hierarchy 3

2.2 Networks and Router Overview 4

2.3 Mesh Networking Overview 5

2.4 DSDV Routing Protocol 7

2.5 Address Resolution Protocol 7

2.6 Wireless 802.11 Standard 8

2.7 Atheros Chipset for Wireless Devices 8

2.8 Gentoo Linux 9

2.9 Linux Kernel Modules 9

2.10 Click Modular Router 9

2.10.1 Elements 10
2.10.2 Operation Mode 10
2.11 Network Analysis Software 11
2.11.1 Ethereal 11
2.11.2 TCPdump 11
2.11.3 Netcat 12
2.12 D-ITG: Dynamic Internet Traffic Generator 12
Chapter 3 - Mesh Router Hardware 13
3.1 Basic Computer System 13
3.2 Selection of Communication Standards 13
3.2.1 Mesh Communication 14
3.2.2 Client Communication 14
3.3 System Hardware Configurations and Integration 15
3.3.1 Packet Forwarding 15
3.3.2 Gateway Router Support 16
3.3.3 Wireless and Wired Client Support 16
Chapter 4 - Software Systems 17
4.1 Gentoo Linux 17
4.2 Madwifi Driver 17
4.3 Orinoco Driver 18
Chapter 5 - Packet Routing: Click Modular Router 19
5.1 Click Elements 19
5.1.1 IPRewriter 19
5.1.2 ICMPPingRewriter 20
5.1.3 ICMPRewriter 20
5.1.4 GatewayDevice: Compound Element 20
5.2 DSDV Implementation 21
5.2.1 Protocol choice 21
5.2.2 Click Implementation 21
5.3 Click Modular Router Configuration Files 23

5.3.1 Packet Forwarding Router Configuration 23
5.3.2 Client Gateway Router Configuration 23
5.3.3 Internet Gateway Router Configuration....24
5.3.4 Block Descriptions....24
Chapter 6 - System Automation 32
6.1.1 Meshstart....32
6.1.2 Memwatch....33
6.1.3 Logging 34
6.1.4 System Replication....34
Chapter 7 - System Integration 36
7.1 Mesh Network Setup....36
7.2 Client Setup 36
7.2.1 Wireless Interface Configuration 36
7.2.2 Wired Interface Configuration 37
7.3 Complete System....37
Chapter 8 - Testing Methodology and Analysis 39
8.1 Preliminary Internet Test 39
8.2 Effect of Number of Hops on the Network....39
8.3 System Load....41
8.4 Fairness....41
8.5 Results and Analysis 42
Chapter 9 - Future Work and Conclusion 43
Chapter 10 - References 45
Appendix A - Click Elements....46
Appendix B - Click Modular Router Configuration Files 48
Appendix C - Automated Startup Code....71
Appendix D - Click Modular Router Output....78
Appendix E - Budget....79

Appendix F - Raw Testing Output 80

V. List of Figures

Figure 2-1: Network Hierarchy 3
Figure 2-2: Generalized Packet Example 4
Figure 2-3: Example of NAT 5
Figure 2-4: Example of Wireless Mesh Network 6
Figure 2-5: Example of Common Wireless Network 6
Figure 2-6: Basic Packet Flow Through Click 10
Figure 2-7: Click Configuration Diagrams: Linux Module and User Level 11
Figure 3-1: Hardware Configuration of Packet Forwarding Router 15
Figure 3-2: Hardware Configuration of Gateway Router 16
Figure 3-3: Hardware Configuration of the Client Gateway Router 16
Figure 5-1: Gateway Device Interaction Diagram 20
Figure 5-2: Block Diagram of Packet Forwarding Router Configuration 23
Figure 5-3: Block Diagram of Gateway Router Configuration 24
Figure 5-4: Directed Graph Showing the Rewriter Block 25
Figure 5-5: Directed Graph of Mesh Wireless Device Input Processing and Packet Classification Block 26
Figure 5-6: Directed Graph of Mesh Wireless Device Output Processing Block 28
Figure 5-7: Directed Graph of Router Input Processing and Packet Classification Block 29
Figure 5-8: Directed Graph of Router Output Processing Block 29
Figure 5-9: Directed Graph of Gateway Device Input Processing and Packet Classification Block 30
Figure 5-10: Directed Graph of Gateway Device Output Processing 31
Figure 7-1: Block Diagram of the Complete Wireless Mesh Network 38
Figure 7-2: Photo of Complete Wireless Mesh Systems 38
Figure 8-1: Basic Linear Topology 39
Figure 8-2: Fairness Test Topology 42

VI. List of Tables

Table 2-1: 802.11 Standards 8
Table 3-1: Hardware Configuration of Mesh Routers 13
Table 6-1: Mesh Start Beep Sequences 33
Table 6-2: Machine Cloning Commands 35
Table 7-1:Commands to Required to Allow Client Connection to Mesh Network 37
Table 8-1: Testing Parameters for Number of Hops 40
Table 8-2: Averaged Collected Data from No. of Hops Tests 41

VII. List of Definitions and Acronyms

DNS - Domain name service is a lookup service that maps human readable hostnames to Internet protocol addresses

DSDV - Destination Sequenced Distance-Vector Routing. A proactive ad-hoc routing protocol.

ESSID - An identifier for a wireless network, used to associate systems attempting to form, or to join, a wireless network.

Ethernet - Packet routing protocol used to route packets in a local area network.

ICMP Packet - Packet sent using the ICMP protocol. This is an IP layer-to-IP layer protocol usually used to send error packets relating to the inter-network routing of a packet.

IP - Internet Protocol. Used for inter-network routing.

Linux Kernel Module - A modular driver that interacts directly with the Linux Kernel. It is the interface between a hardware device or a software program and the Linux Kernel.

MAC Address - The unique hardware address assigned to each network device by the manufacturer. It is used by the Ethernet protocol.

Memory - The memory, or RAM, of a system refers to the dynamic storage space used to temporarily store programs and their associated data while they are running.

Network Address Translation (NAT) - Source address translation done in order to allow packets from many systems on one network to flow through a router to their destination on another network. Packets that arrive at the router have their source address changed to be that of the router, and are sent out to another network. When the replies arrive, the destination addresses are translated back to match that of the original sender.

Open Source - Software whose source code is publicly available to be modified.

Packet - Fragmented piece of a data stream, routed autonomously from sender to receiver.

PCI - Peripheral Component Interconnect. A standard commonly used to connect peripheral hardware to a Personal Computer.

PCMCIA - A standard commonly used to connect peripheral hardware to a laptop.

Perl - A computer programming language commonly available on UNIX systems. It contains good functions for creating command line scripts.

Round Trip Time (RTT) - The amount of time between a data packet is sent onto the network and a response is received.

TCP - Transmission Control Protocol. A transport layer end-to-end data transmission protocol.

Throughput - The speed at which a computer can send and receive data on a network, is know as throughput. This is generally measured in bit/s

UDP - User Datagram Protocol. A transport layer end-to-end data transmission protocol.

WMN - Wireless Mesh Network.

Chapter 1 - Introduction

1.1 Overview

The goal of this project is to create a working model of a wireless mesh network (WMN) using off the shelf components and to gather data from test patterns to identify the utility of this type of network. In a WMN routers are not connected via a wire to the Internet, as is done in the current wireless networking system. Instead, data must be routed by forwarding packets via wireless communication from clients through mesh routers to a gateway router that is connected to the Internet. This type of network is known as a multi-hop network, since data must hop from one router to another in order to reach its destination. In order to create such a network, appropriate hardware must be chosen. Software must then be installed on the routers to support this hardware, as well as to support their functionality. Finally, a routing program must be created to define the packet flow through the network.

1.2 Purpose

The goals of the project are as follows:

- To reduce infrastructure costs for new networks.
- To create an easily deployable network.
- To design a multi-hop network to support standard user internet traffic.
- To test the effect of the number of hops and the load produced by the number of clients on the performance of the designed network.

1.3 Scope

The intention of this project was to create a network that would handle standard internet user traffic, including web browsing, e-mail and file transfers. Clients should be able to connect to one mesh router at the end of the network, and have their data passed along to a gateway mesh router, to access the Internet. The routers should use common consumer hardware, and existing network protocols. Analysis should be preformed on the mesh network based on the number of hops, the amount of traffic on the network and the fairness exhibited between clients on the network. Client mobility, automatic network configuration and wireless security were outside the scope of the project; however these would be interesting topics for extensions of this project.

1.4 Organization of Report

The paper is split into 9 sections. Chapter 2 details background information about the components of the project. Chapter 3 defines the hardware used in the mesh network. Chapter 4 discusses the software configuration needed by the routing software, discussed in Chapter 5. System automation is discussed in Chapter 6. Chapter 7 deals with how these components interact in order to form the mesh network. Chapter 8 discusses the testing of the network, and the results achieved. The conclusion and future work possible for this project are discussed in Chapter 9.

Chapter 2 - Background

This section will provide insight and background to the advanced topics discussed throughout the report.

2.1 Network Layer Hierarchy

As this project deals largely with networking concepts, a discussion of the network hierarchy that data passes through on computer systems needs to be discussed. The network TCP/IP hierarchy can be broken down into four basic logical layers. These are, from top to bottom, the application layer, the transport layer, the network layer and the physical or network interface layer as shown in Figure 2-1. The relevant layers for this project were the transport, network and physical layers.

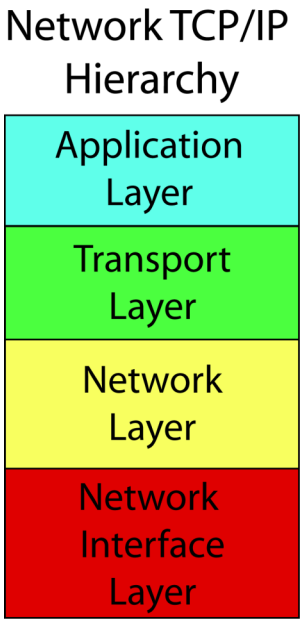


Figure 2-1: Network Hierarchy

The application layer is governed by the programs that generate the data. As this project is not concerned with what applications are being run, this layer will not be discussed.
The transport layer deals with end-to-end data correctness. It is at this layer that data is segmented into packets. Two important protocols at this layer are TCP and UDP. The TCP protocol defines a series of handshaking steps needed to create a connection, and maintains a stable connection by retransmitting dropped packets. The UDP protocol is a best-effort connectionless protocol. It provides no mechanism to establish a connection, or retransmit dropped packets.

The network layer is in charge of inter-network packet transmission. The Internet protocol (IP) is the standard network layer protocol for the Internet. It defines an IP address unique to each system on a network.

The network interface layer is used to transmit packets within a local area network at a hardware and software level. Each network device has a hardware address, or MAC address, unique to it. The Ethernet protocol is used to address packets to specific systems on a local area network. This is done for both wired and wireless networks. In order to keep track of information, each layer adds a header to the data packet. An example of this is shown in Figure 2-2. [1]

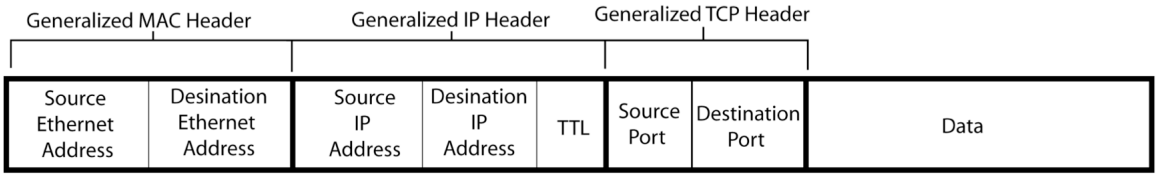


Figure 2-2: Generalized Packet Example

2.2 Networks and Router Overview

Basic networking concepts need to be discussed, as these concepts are important to the project. A network is a collection of computers connected by a physical medium. Most networks contain a small number of systems, and can only access systems local to their network, these are called local area networks(LAN). For these systems to access external resources such as the Internet, a router is needed. Routers act as a junction between two networks, transferring data packets between them. As the addressing of these networks can be different, translations are needed, and are performed by the router in a process called network address translation (NAT) as shown here in Figure 2-3. [2]

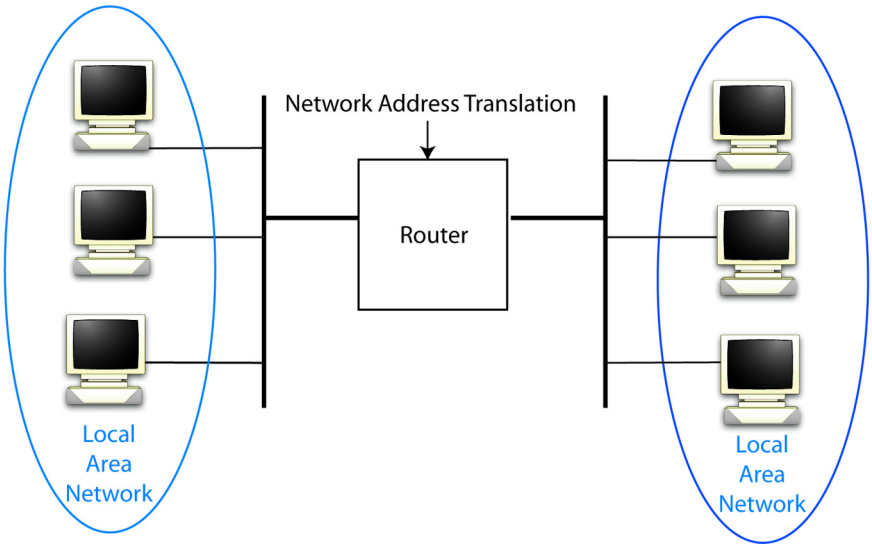


Figure 2-3: Example of NAT

To access resources outside its network a system sends data to the router, which passes the packet along to the outside network. Packets that arrive at the router from a system have their source address changed to be that of the router, and are sent out to another network. When the replies arrive back at the router, the destination addresses are translated back to match that of the original sender. This process of packet flow management is known as routing. As can be seen in the figure below, a computer on network A sends data through the router to a computer on network B. The source and destination address of the data packet changes as the router performs NAT.

2.3 Mesh Networking Overview

Figure 2-4 shows the overall topology of a wireless mesh network. Wireless mesh networks are networks where connections are formed between each router rather than connecting through a central gateway. These networks are commonly known as multi-hop networks, as a packet must hop wirelessly a number of times from one router to the next before they reaching the gateway.

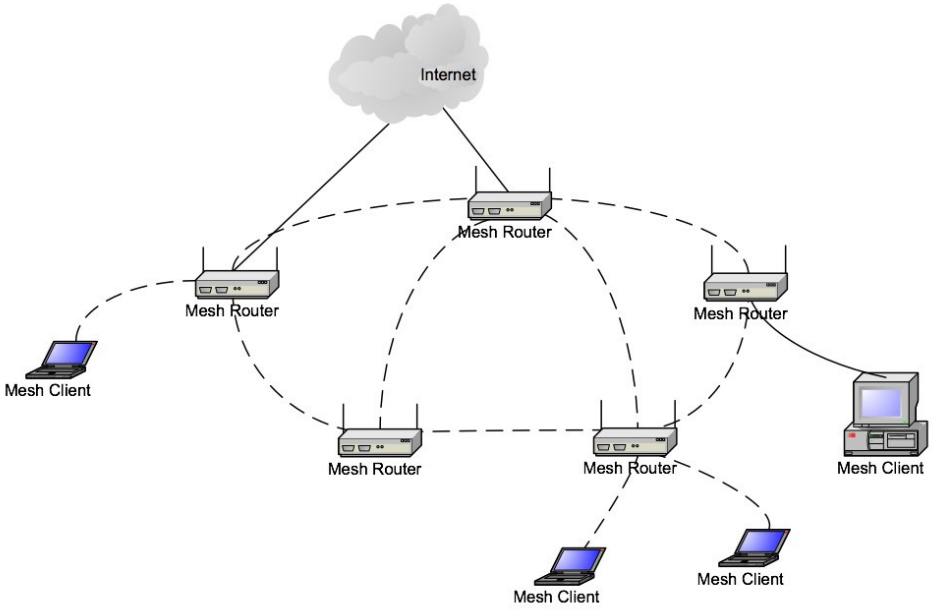


Figure 2-4: Example of Wireless Mesh Network

This is different from the current system shown in Figure 2-5, where a packet will only ever pass through one wireless hop before reaching the gateway. Note that in both diagrams the gateway routers are connected to the Internet, rather than another external network.

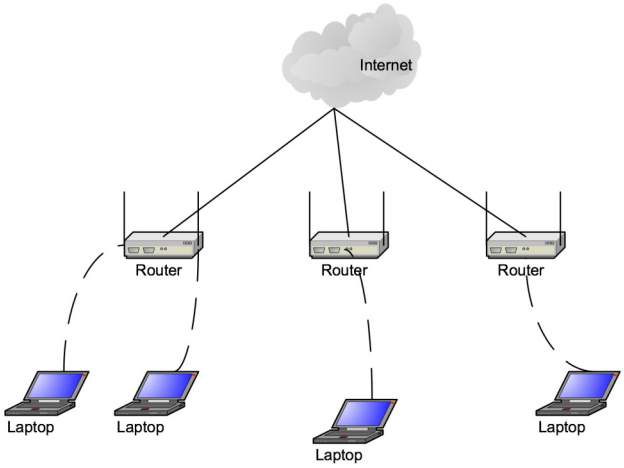


Figure 2-5: Example of Common Wireless Network

In wireless mesh networks there is a performance cost attributed to the network since each packet must pass through a number of routers to reach its final destination. This is an expected trend due to the fact delays occur at each hop and the more hops a packet must traverse the more delay it will experience.

Wireless mesh networks can be broken down into two broad architecture categories, infrastructure mesh networks and client mesh networks. Infrastructure wireless mesh networks allow regular clients to connect to a mesh backbone that takes care of the packet routing. This routing is transparent to the client. Client mesh networks involve the clients in the routing process. Packets may be forwarded to a client or a mesh router, and the devices are expected to forward the packet as necessary. In this architecture, clients must run specialized routing software, and give up system resources to maintain the mesh network [3].

2.4 DSDV Routing Protocol

There are a number of multi-hop routing protocols for mesh networks. One such protocol is Destination-Sequenced Distance-Vector (DSDV) Routing, which is used in the project. Multi-hop routing protocols can be separated into two broad categories: reactive routing protocols and proactive routing protocols. Reactive protocols will discover route information dynamically, whenever a packet arrives and no current route information is available. Proactive protocols, such as DSDV, keep track of routing information through periodic updates, so that it is always available. [4]

Under the DSDV routing protocol, every host will maintain distance estimations to every other host participating in the network. This distance is measured in the number of hops needed to reach a given host. The route information is determined through the broadcast of route advertisement packets to all hosts within communication range.

There are two types of broadcasts. The full dump broadcast occurs periodically, and sends all entries from the host's route table to all adjacent hosts. The other type is an incremental advertisement. This is sent between full dump broadcasts, if a large change in routing information has occurred. For example, if a host has gone offline, an incremental advertisement would be made.

Routes are based on the age of the route information, and on the shortest path available to the desired host. Packets received by a host that are addressed to another host in the network are forwarded along to the next hop, until the packet reaches its destination. Gateway hosts connected to external networks are flagged and packets destined for systems outside of the network are forwarded to the closest gateway host. [4]

2.5 Address Resolution Protocol

The Address Resolution Protocol (ARP) is a protocol used to discover the MAC address of a system given its IP address. A system that wants to know a MAC address will broadcast an ARP

request containing the IP address of the desired system. The destination system will recognize the request, and send a reply to the requesting system with its MAC address. The requesting system can then send other packets directly to the destination system by encapsulating them in an Ethernet header containing the destination system's MAC address. [1]

2.6 Wireless 802.11 Standard

The 802.11 Wireless Networking Protocol defines how Ethernet packets are sent between 802.11 devices. It operates at the network interface layer and includes three main standards. The standards commonly in use are: 802.11a, 802.11b and 802.11g and are shown in Table 2-1.

Table 2-1: 802.11 Standards

Standard	Center Frequency	Non-Overlapping Channels	Max. Transmission Rate
802.11a	5 Ghz	11	54 Mbps
802.11g	2.4 Ghz	3	54 Mbps
802.11b	2.4 Ghz	3	11 Mbps

802.11a operates in the 5 GHz range and has eleven non-overlapping channels so that adjacent networks do not interfere with one another. The high operating frequency limits the range and penetration of the network's signals, but the network avoids interference with devices operating in the 2.4 GHz range. 802.11a networks can send data at a maximum rate of 54 Mbps.
The 802.11b and 802.11g networks both operate at the 2.4 GHz frequency. They differ in how they modulate the data in the wireless medium. 802.11b networks use Direct Sequence Spread Spectrum modulation, and can send data at a maximum rate of 11 Mbps. 802.11g networks use orthogonal frequency division multiplexing, and can send data at a maximum rate of 54 Mbps. Due to the lower operating frequency of these two standards, the range and penetration of the network signals are greater than 802.11a. [2]

2.7 Atheros Chipset for Wireless Devices

These chip sets are commonly found in networking devices that support the 802.11a and 802.11g modes, including models from D-Link, Linksys and Netgear. Because of their popularity among wireless card manufactures we determined that devices using these chipsets would work well for the project.

2.8 Gentoo Linux

Gentoo is an operating system based on the open source Linux kernel. A unique feature of Gentoo is the way that it distributes its software. Instead of precompiled binaries, Gentoo uses a system called Portage that defines the dependencies of a particular software package, where to download the source code and how to configure and build it locally on the machine. This allows for the software to be customized for the particular hardware it is being run on. Another benefit of Gentoo Linux is that its installation size is small, and only includes what is needed to boot the machine and some basic system utilities. All other software must be acquired through Portage.

2.9 Linux Kernel Modules

A feature of the Linux kernel is its ability to load object files called kernel modules that contain code to add features and functionality to the already running kernel. This increases the scalability of the system, and when the kernel module is not needed any more it can be unloaded, freeing system memory. This project deals with a number of kernel modules that will be discussed in the relevant sections.

2.10 Click Modular Router

The Click Modular Router (Click) is a packet routing software package [5]. It allows the configuration of a standard computer system into a fully functional router. Click defines what happens to a packet when it arrives at a router. The packet must be processed before it leaves the router in order to perform routing operations such as network address translation and to address the packet using the protocol of the destination network. It also has the ability to classify a packet, for example it can classify a packet by destination in order to determine which network the packet should be output on. This is summarized in Figure 2-6, which shows a directed graph of the flow of a packet through a Click router.

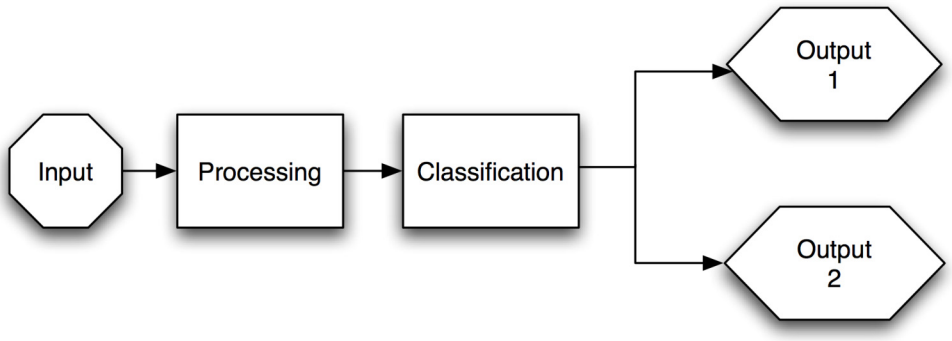


Figure 2-6: Basic Packet Flow Through Click

2.10.1 Elements

In order to implement packet routing, Click breaks the processing and packet classification operations down into elements, which are defined in a configuration file. This configuration file is what defines the operation of a Click router. The programming language for the configuration file uses declaration constructs to define which elements to include in the router and connection constructs to define how packets flow between them. This creates a directed graph similar to the one shown above, where packets flow along the edges to the elements at the vertices for processing.

In some instances information about a packet that is determined by one element is needed by another at a later time. This information is added to the annotation space in the packet's header. It is ignored once the packet leaves the click router. This is used most often to set a packet's destination IP address.

The click programming language allows the creation of compound elements. This is a collection of connected elements that can be viewed as one element from the outside. They will accept input, process the packets in some way, and produce them at an output. This simplifies the configuration file, as the elements in the middle of the compound element do not need to be declared in advance.

2.10.2 Operation Mode

The Click Modular Router can be run in two different modes under the Linux operating system, as a user-level driver, or as a kernel module. As a user-level driver it receives packets after they have been handled by the Linux kernel. As a kernel module, it allows the router to receive packets before they are processed by the Linux kernel, and thus packets can be redirected without

the Linux kernel ever seeing them. The efficiency of the kernel module is higher than user-level driver as the kernel does not have to process the packet before Click does. The configuration files for the user-level driver cannot be used to define a kernel configuration, or the other way around. The kernel module and user level configuration diagrams can be seen in Figure 2-7

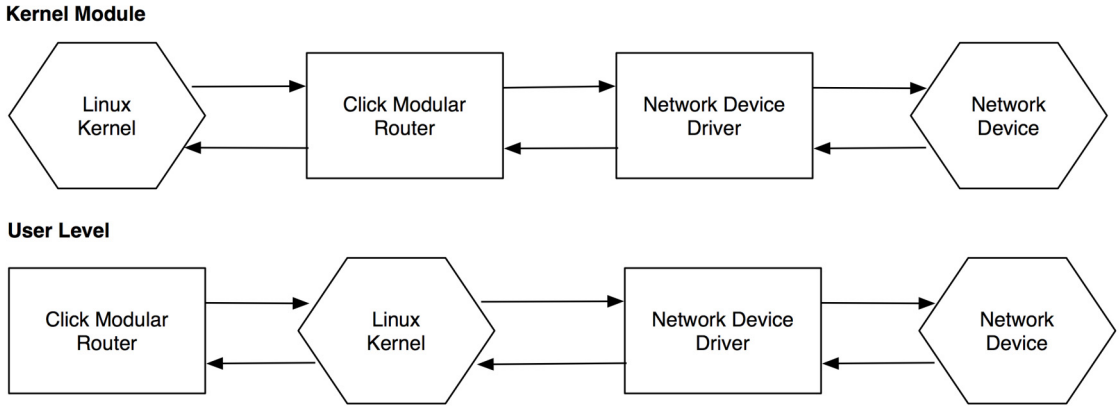


Figure 2-7: Click Configuration Diagrams: Linux Module and User Level

2.11 Network Analysis Software

There was a number of network analysis programs used to debug the mesh network. These included packet capture programs such as Ethereal and TCPdump, and a network utility called Netcat.

2.11.1 Ethereal

Ethereal is a common network packet capture program used to monitor network traffic. It provides a graphical user interface, and has features to analyze network data such as filtering capabilities, and graphing of throughput and round trip time. It also has the ability to group packet data based on the protocol of the packet.

2.11.2 TCPdump

TCPdump is a network packet capture program. It allows the capture of data coming into and out of the mesh routers. Problems with network communication can easily be debugged with TCPdump. This tool does not require a graphical user interface, and can be run on a remote system.

2.11.3 Netcat

Netcat is a networking utility that allows data to be read and written to TCP connections. It can be used for network debugging and analysis. It can connect to a remote system on a specified port, and pass commands and read information from the software listening on that port

2.12 D-ITG: Dynamic Internet Traffic Generator

The D-ITG program that generates data packets to simulate network traffic. It is able to create data packets using the ICMP, UDP and TCP protocols. The size of the packets, interval between the packets, length of test, and packet rate can also be controlled. These parameters can either be given a constant value, or specified to follow a random distribution. At the end of each simulation a numerical summary of the data flow is provided. [6]

Chapter 3 - Mesh Router Hardware

This section outlines the hardware specifications, components and configurations that make up the mesh routers for this project. The various hardware configurations allowed communication between each of the wireless mesh routers, between a router and its clients, and between a router and the Internet. The hardware design decisions will be described in the first two sections of this chapter. The third section shows how it was integrated to form the base of a wireless mesh router.

3.1 Basic Computer System

The Basic Computer system allowed integration of multiple PCI adapters supporting wireless and wired Ethernet communication. In addition to supporting multiple PCI adapters the processing power of the systems also supported a minimal Unix based operating system and were able to route packets from multiple sources efficiently with out lag due to system load. Table 3-1 shows the specifications and components of the system. Although the system's speed was below the standards of a common desktop, they were more than adequate in processing power for the project. Small integrated system boards were considered but exceeded the allowed budget and lacked the expandability to be a complete solution. The system chosen had three PCI slots, which allowed for flexibility in regards to the communication device configuration.

Table 3-1: Hardware Configuration of Mesh Routers

Hardware Configuration of Mesh Routers
CPU: Intel Pentium II processor running at 200 MHz
Memory:64 MB RAM
HD: 3 GB hard disk drive
Video: ATI Rage Pro AGP 1X/2X Video Card
Network: D-Link DWL-AG530 Atheros based wireless Ethernet card.
Realtek Semiconductor wired Ethernet card, installed in two machines.
U.S. Robotics USR2410 Wireless PCI Card, installed in one machine

3.2 Selection of Communication Standards

Wireless communication between mesh routers and clients was a requirement of the project. The 802.11 communication standard was selected as the medium for communication since it is

commonly found in consumer hardware, easy to acquire, and operates on unregulated frequency bands. Its large range of communication channels made it easy to use and expandable.
This project implements an infrastructure wireless mesh network. This was implemented so that clients could connect to the network without requiring them to have special software or route packets. Not requiring client to have special software improves overall accessibly to the network. As well, the client's system resources are not diminished and network stability will not be degraded by clients entering and exiting the network.

The 802.11a standard was well suited for the mesh side of the network where as 802.11b\g was suited for the client side. This combination of an infrastructures network for the mesh and having the clients on separate frequency bands provided for optimal future expandability and prevented any interference between the two interfaces.

3.2.1 Mesh Communication

The mesh communication system was the link between each router, implemented through the 802.11a standard. This communication standard had to meet many design specifications. It must have the highest data rate available for consumer level wireless communication. This layer must suffer minimal interference from other devices, as it is the backbone to the network. The 5GHz operating frequency of 802.11a is best suited for this since it is protected from interference common at the 2.4 GHz range. For example, the existing university network is an 802.11g network and interference with this system had to be avoided.
The mesh communication system used an arbitrary channel of 30, chosen simply because it was in the middle of the available channels. The D-link AG-530 card met all these design requirements for wireless 802.11a communication. It was also know to be compatible with the Linux operating system, since it used an Atheros chipset. The compatibility and driver will be discussed in Chapter 4.

3.2.2 Client Communication

The client communication required a commonly used standard to allow ease of connections for nearly all clients at a high data rate. This could be done wirelessly though the 802.11b or 802.11g standards, since they are both commonly used for clients accessing a wireless network. Choosing one of the above standards would also guarantee that clients would not interfere with the mesh communication. In the network created, 802.11b was used for wireless communication between clients and mesh routers. Although this protocol did not allow the same data rate as 802.11g, it was well supported under the Linux operating system. A card was available for use and

demonstration purposes, so wireless communication could easily be added without the risk of using unsupported hardware. Also, there were reports that machines could not be configured with two cards both containing Atheros chipsets. Since the most common chipsets for cards implementing the wireless 802.11g standard is the Atheros chipset, and this information is not advertised on the product, the 802.11b standard was more attractive. The low data rate did not adversely affect the network, since wired connections were used for testing purposes.

3.3 System Hardware Configurations and Integration

Not all routers require the same hardware configuration in the network created. Some would need to connect to external networks, whereas others would communicate only with other mesh routers. Although the systems were designed to be similar there were variations in hardware configurations done in order to meet these specific requirements. All configurations contain the D-Link DWL-AG530 802.11a cards to perform mesh communication. The different hardware configurations were: Packet Forwarding, Gateway Router Support, Wired and Wireless Client Support.

3.3.1 Packet Forwarding

The first configuration was the most basic configuration and is referred to as a packet forwarding router in the report. These machines had one D-Link DWL-AG530 802.11a wireless card. The single purpose of these routers was to forward packets from one mesh router to. These routers looked at a packet's destination, as described Chapter 5, and determined the path the packet needed to take. These routers were found within the mesh network, and did not have the hardware or configuration to be connected to the Internet or to allow clients to connect. Figure 3-1 below shows the configuration of one of these devices.

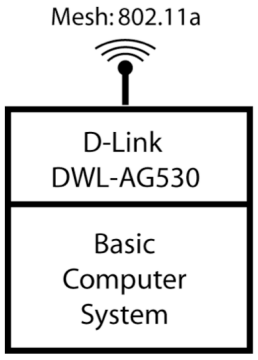


Figure 3-1: Hardware Configuration of Packet Forwarding Router

3.3.2 Gateway Router Support

The second type of mesh router was an Internet gateway router. These routers contained two network devices, one D-Link DWL-AG530 802.11a wireless card to connect to the mesh network, and a wired Ethernet card to connect to the Internet. These mesh routers were configured as gateways, and, as described in Chapter 5, advertised themselves as such so that other mesh routers would know where to forward packets destined to an external network. The Figure 3-2 below shows how one of these devices is configured.

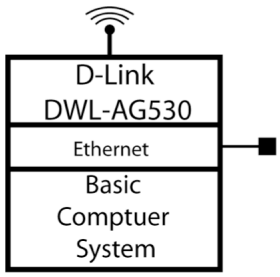


Figure 3-2: Hardware Configuration of Gateway Router

3.3.3 Wireless and Wired Client Support

The last type of mesh router was a client gateway router. These routers contained two network devices, one D-Link DWL-AG530 802.11a wireless card to connect to the mesh network, and a second network card to connect clients to the mesh network. The second network card was either through a wireless network shown in Figure 3-3 or a wired network with the same hardware configuration as Figure 3-2, depending on the type of clients that were connected. For this project, there were two machines that could be configured as a client gateway, one with a wired Ethernet adapter and another with a U.S. Robotics USR2410 802.11b wireless card.

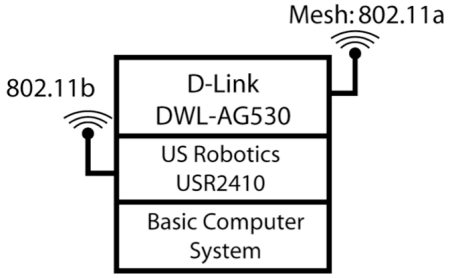


Figure 3-3: Hardware Configuration of the Client Gateway Router

Chapter 4 - Software Systems

This chapter discusses software design decisions in terms of the Gentoo Linux operating system, the Madwifi Driver for Atheros based devices, Orinoco Driver used for the wireless client interface, and the system automation scripts.

4.1 Gentoo Linux

The operating system was required to support the Click Modular Router and allow our chosen wireless cards to work. Since the Click Modular Router only runs on Linux and BSD variants, we were limited to these operating systems. There also needed to be support in the form of a driver to allow the D-Link DWL-AG530 wireless cards to function.

There are a number of different Linux distributions available. A distribution that was not overly specialized was required. However, the distribution could not be too complex, as it would overwhelm the systems being used. Distributions specific to routing applications, such as Clark Connect and the Linux Router Project were researched. It was found that most of these distributions were meant to be installed and configured based on the distributions specifications. This would have limited the ability to use other software, and configure these distributions to meet the projects specifications. Other distributions, such as Fedora and OpenSuse were also considered. These are very popular and flexible distributions; however they are generally used on systems ranging from modern desktops to servers, and require more computing power than the machines for this project had.

Once the Gentoo operating system was installed, only the software needed to boot the machine and some basic utilities were available. When any other software was needed, it was downloaded as source code and compiled on the system itself using the Portage system. This allowed only the software that was needed for this project to be installed, including support utilities for the router software and diagnostic software to monitor and debug the system. The systems were therefore configured with a minimum installation, while still remaining very flexible.

4.2 Madwifi Driver

As mentioned in Chapter 3, the wireless cards that were used were the D-Link DWL-AG530's based on the Atheros chipset. Under Linux these devices are supported with a Kernel Driver from the Madwifi project. There are two parts of this driver, the ath_pci and the ath_hal kernel modules. Atheros devices are able to change their channels to ones not defined for the 802.11 protocols. To limit a user to the 802.11 standard channels, Atheros provides a hardware

abstraction layer (HAL) module that is called ath_hal. The ath_pci module allows the kernel and networking utilities to interact with the device, and uses the ath_hal module to interact with the device itself. The Madwifi driver is open sources, however the HAL is not, as that would allow users to modify it to access frequency ranges not defined in the 802.11 standards.

The Madwifi driver worked well for this project, as it was able to go into ad-hoc mode, and could be used in both 802.11g, for debugging purposes, and 802.11a channels. By using this driver the mesh network was assembled as proposed. However, when testing it was discovered that the mesh routers were running out of memory and crashing. This was traced back to the Madwifi driver, to a problem that had been reported on the Madwifi website on February 9th, 2006 [At the time of writing, the report was available at: http://madwifi.org/ticket/377]. Testing created a high amount of traffic, much more than regular web or email traffic generated by one or two users, and this was causing the mesh routers to crash. Due to this problem, the testing was not completed as proposed. Please see Chapter 7 for more details.

Another option that would have allowed the wireless cards to work under the Linux operating system was the ndiswrapper module. This module is a wrapper for windows drivers, that allows them to be used under Linux. As this is a wrapper, the performance compared to the madwifi driver would have been much slower.

4.3 Orinoco Driver

To support the U.S. Robotics USR2410 wireless card, the Orinoco driver was used. This driver supports Prism based wireless cards, including the U.S. Robotics card used in this project. The wireless card and this driver were installed and configured in the Client Gateway Router to allow wireless clients access to the mesh network.

Chapter 5 - Packet Routing: Click Modular Router

The Click Modular Router was responsible for defining how each mesh router would route packets throughout the system. Packets are sent from Click to the network driver, a communication device connected to the host router, or the Linux kernel of the host system its self. Designed to be flexible and configurable, Click uses a number of routing elements each performing a different operation on a packet. Packets flow through these elements, eventually reaching their output destination.

Section one of this chapter will discuss the important Click elements used in the project's router configuration files. The implementation of the DSDV Routing Protocol and the relevant click elements used in its configuration are presented in section two. The packet flow through the complete routing system implementing both Click and DSDV elements is shown in section 3.

5.1 Click Elements

The following section will discuss four configurable Click elements used in the project's router configuration files. The processing preformed by these elements is described in the following sections. Most elements did not require specific configuration. For a description of the other elements included in the configuration files, see Appendix A.

5.1.1 IPRewriter

This element was used to perform NAT on TCP and UDP packets. This section describes the address translation scheme used by the IPRewriter. Then, the addressing patterns used in the network are described. The reverse flow case is shown, and then finally the inputs and outputs of the element are described.

The element created a mapping based on a packet flow identifier. This was a set containing the source address, source port, destination address, destination port and protocol of the packet. The packet flows were mapped based on patterns to change source IP address and source port of the packet requiring network address translation. The IPRewriter elements implemented in the configurations used in the mesh network had the same format for each pattern. They changed the source IP address to be that of the host router, and assigned the flow a source port on that router in the range of 50000-65535. The destination IP address and port were left unchanged. A special pattern named 'pass' was also used; it did not modify any parameters of the flow identifier. The IPRewriter element kept track of the packet flows for packets that were input. If no previous mapping existed, a new one was created. If a previous mapping existed, the packet's addresses

were translated using that flow mapping. Packets would also be mapped based on if they matched the reverse of that mapping, in order to return packets to their original sender.
Each pattern in the IPRewriter element had a corresponding input. For example, the first pattern would accept packets on the first input. After the pattern, an output was defined for the forward and reverse paths. For simplicity, in the configuration files created, the output for forward path of the first input was the first output, and so forth. The reverse output would correspond to the forward output of the other mapping pattern.

5.1.2 ICMPPingRewriter

This element is similar to the IPRewriter, but preformed NAT only on ICMP echo packets. It did not have the same amount of flexibility as the IPRewriter. It was configured to change the source address to correspond with that of the host router, and leave the destination address unchanged. Requests were sent to the first output, replies were sent to the second output.

5.1.3 ICMPRewriter

This element was used to check mappings for ICMP error packets. It checked the patterns used in the IPRewriter and ICMPPingRewriter elements. If a mapping existed, it was applied, and the packet sent to output zero. If not, the packet was sent unchanged to output one.

5.1.4 GatewayDevice: Compound Element

This element manages packet flow between Click and a communication device. In the network implemented, this communication device was the client wireless device or wired Ethernet device. As shown in Figure 5-1, packets coming from the device were sent through to the next element within the Click router. Packets destined for the external network were first sent to a queue, from the queue packets were sent to the device. Since this element could not write to and read from the device at the same time a schedule was created. The schedule was such that the packets where sent to the device ten times more often than read from the device.

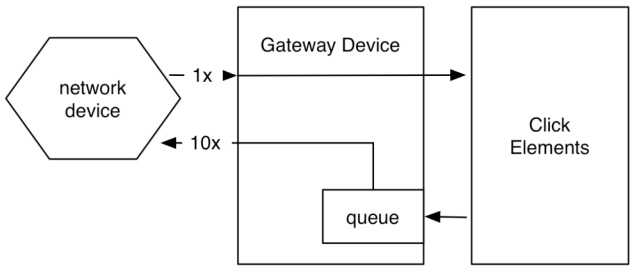


Figure 5-1: Gateway Device Interaction Diagram

5.2 DSDV Implementation

In order to route packets between mesh routers, a networking protocol had to be chosen. In the network created, the DSDV protocol was used. The benefits of this protocol are described in section 5.2.1. The implementation of this protocol within Click is described in section 5.2.2.

5.2.1 Protocol choice

The DSDV routing protocol was chosen because it was designed to work with a wireless network of routers that know nothing about the other routers in the network. It takes advantage of the wireless medium by using broadcast packets for route advertisement packets, but does not flood the network with these packets. [4]

The DSDV routing protocol has been well tested in the past, allowing results from our network to be applied to other networks. Because of the configuration of the mesh network created, a proactive protocol, such as DSDV, was better suited to it. This is shown in two studies. In one study comparing DSDV to AODV, a reactive protocol, it was found that DSDV preformed better with higher traffic loads [7]. Another study also indicated that DSDV performs well when nodes are not very mobile, dropping very few packets with a constant overhead of routing packets as the number of routers increased. [8]

5.2.2 Click Implementation

The DSDV elements are part of the grid package of the Click Modular Router. They were created to implement the DSDV Protocol as described in [4]. The following four elements were used in the routing configuration files used in the mesh network.

5.2.2.1. GridGatewayInfo

This element had two purposes. It defined if a router was a gateway, and addressed packets to the closest gateway. If a router was acting as an Internet gateway, the element was declared with the gateway flag set to true. If not, the flag was set to false. The element would change the destination IP address annotation of an incoming packet to be that of the closest gateway, chosen from the DSDVRouteTable element discussed below. If a router was a gateway, packets were not sent through the GridGatewayInfo element, since they were to be forwarded out of the mesh network by this router.

5.2.2.2. DSDVRouteTable

This element contained the routing table for the host router. It kept track of all of the information known about the other routers in the network. The element kept track of such information as whether the router was a gateway, if the router was online, the IP address of the router, the IP address of the next hop router, and the number of hops needed to reach a router. The information came from route advertisement packets. This element also periodically generated these route advertisement packets.

5.2.2.3. GridEncap

This element encapsulated a packet in a grid header. The grid header contained protocol specific information, such as the source and next hop IP and MAC addresses of the packet. As well, information regarding the purpose of the packet was included in the header. The two purposes used in the mesh network were data packets and route advertisement packets.

5.2.2.4. FromGridDev - Compound Element

All packets entering the router from the mesh wireless device were processed by this compound element. All packets were first sent to a Classifier, which sent the packets to different outputs based on their type. Packets with a grid header were sent to output zero. Packets containing linkstat statistical information would have been sent to output one. Linkstat information was not implemented in this mesh network. Packets that were not matched were sent on output two.

5.2.2.5. ToGridDev - Compound Element

All packets leaving a router on the mesh wireless device went through this compound element. In the mesh network, two types of packets were implemented, route advertisement packets and data packets. They were classified and treated differently. A priority queue was created to manage packets going to the wireless device. Route advertisement packets had their grid header source address set. They were then given a higher priority in a priority queue.

Data packets had their destination address in their grid header set to the next hop required in order to reach the actual destination. The next hop address was found from the DSDVRouteTable. The data packets then had their grid header source address set to the current router's address, and were sent to the priority queue with a lower priority than the route advertisement packets. If the destination was not in the DSDVRouteTable, the packet was dropped.

5.3 Click Modular Router Configuration Files

There were three different Click Modular Router configuration files used in this project, corresponding to the three different hardware configurations presented in Chapter 3. The configurations defined the actions of a packet forwarding router, a client gateway router, and an Internet gateway router. The actions described in the packet forwarding router configuration are also implemented in the other two configurations. The configurations have been broken down into logical blocks. These blocks will be described in detail in section 5.3.4. The Click Modular Router code is governed by a license, available in Appendix B. Complete diagrams showing how all the elements are connected in the three configurations, along with the configuration files themselves can also be found in Appendix B.

5.3.1 Packet Forwarding Router Configuration

This configuration file was very similar to the simple-dsdv.click sample configuration file available with the Click Modular Router software package[1]. It was modified to suite the application presented in this project. It expected the machine to have one wireless device. Packets would be received on the D-Link DWL-AG530 device, and either kept if they were addressed to the current router or forwarded to the appropriate router. If a packet was destined for an outside network, it would be forwarded to the gateway router. The configuration can be broken down into four main packet processing sections, as can be seen in Figure 5-2.

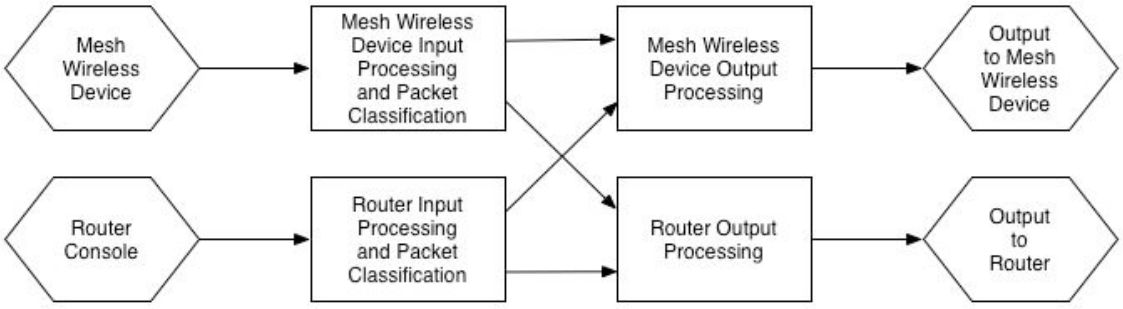

Mesh Wireless Device
Mesh Wireless Device Input Processing and Packet Classification
Mesh Wireless Device Output Processing
Output to Mesh Wireless Device
Router Console
Router Input Processing and Packet Classification
Router Output Processing
Output to Router

Figure 5-2: Block Diagram of Packet Forwarding Router Configuration

5.3.2 Client Gateway Router Configuration

This configuration expected the machine to have two devices, one of which should be the D-Link DWL-AG530 card wireless device used to participate in the mesh network. The client device

[1] Click Modular Router available at http://pdos.csail.mit.edu/click/

could have been another wireless or an Ethernet card. We tested our network with both the U.S. Robotics USR2410 and the Realtek Ethernet card. Please see Section 3.2 for more details. This configuration preformed network address translation (NAT) on the packets in order to pass them from one device to the other. Packets not addressed to the host would be forwarded to the appropriate router. The operation of this configuration can be broken down into 7 main parts, as can be seen in Figure 5-3.

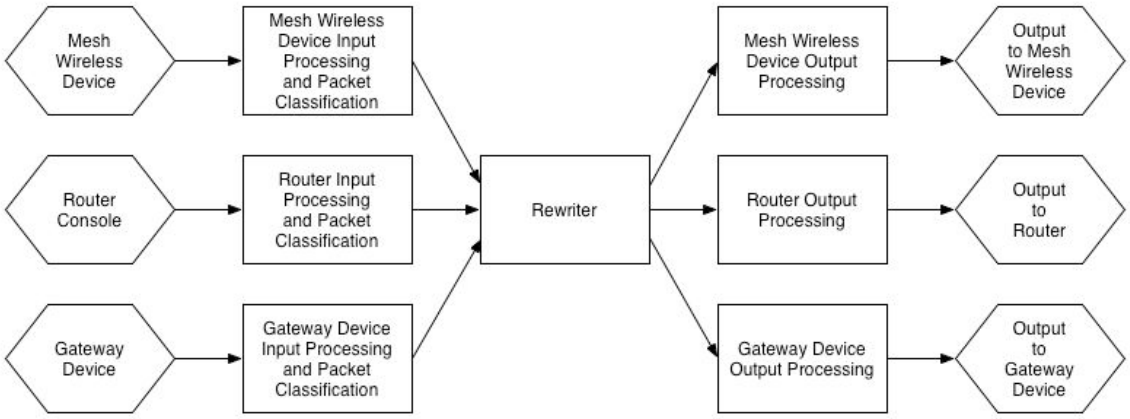

Mesh Wireless Device
Mesh Wireless Device Input Processing and Packet Classification
Router Console
Router Input Processing and Packet Classification
Gateway Device
Gateway Device Input Processing and Packet Classification
Rewriter
Mesh Wireless Device Output Processing
Router Output Processing
Gateway Device Output Processing
Output to Mesh Wireless Device
Output to Router
Output to Gateway Device

Figure 5-3: Block Diagram of Gateway Router Configuration

5.3.3 Internet Gateway Router Configuration

Similar to the Client Gateway Router Configuration, this configuration expected the machine to have two devices, one of which was the wireless device used to participate in the mesh network. The other device was an Ethernet adapter. In the mesh network created, the DWL-AG530 card was used for as the mesh wireless communication device, and the Realtek Ethernet card was used to send packets to the external network. Please see Section 3.2 for more details. In order to pass packets from one device to the other, network address translation (NAT) was preformed. The operation of this configuration can be broken down into the same 7 main blocks, as the Client Gateway Router Configuration was. Please see Figure 5-3 for more details about the interaction between the blocks.

5.3.4 Block Descriptions

In this section a detailed description of each main block of the above configurations is given. All of the blocks had similar functionality in all the configurations. If a major difference in functionality between two configurations existed, it is noted in the block descriptions. In the diagrams, the different paths are contained in one box.

5.3.4.1. Rewriter Packet Flow

The rewriter elements preformed the packet classification for the packets input into the gateway routers. For a description of how the rewriter elements were configured, please see Section 5.1. The simplified digraph of the elements used to do this can be seen in Figure 5-4.

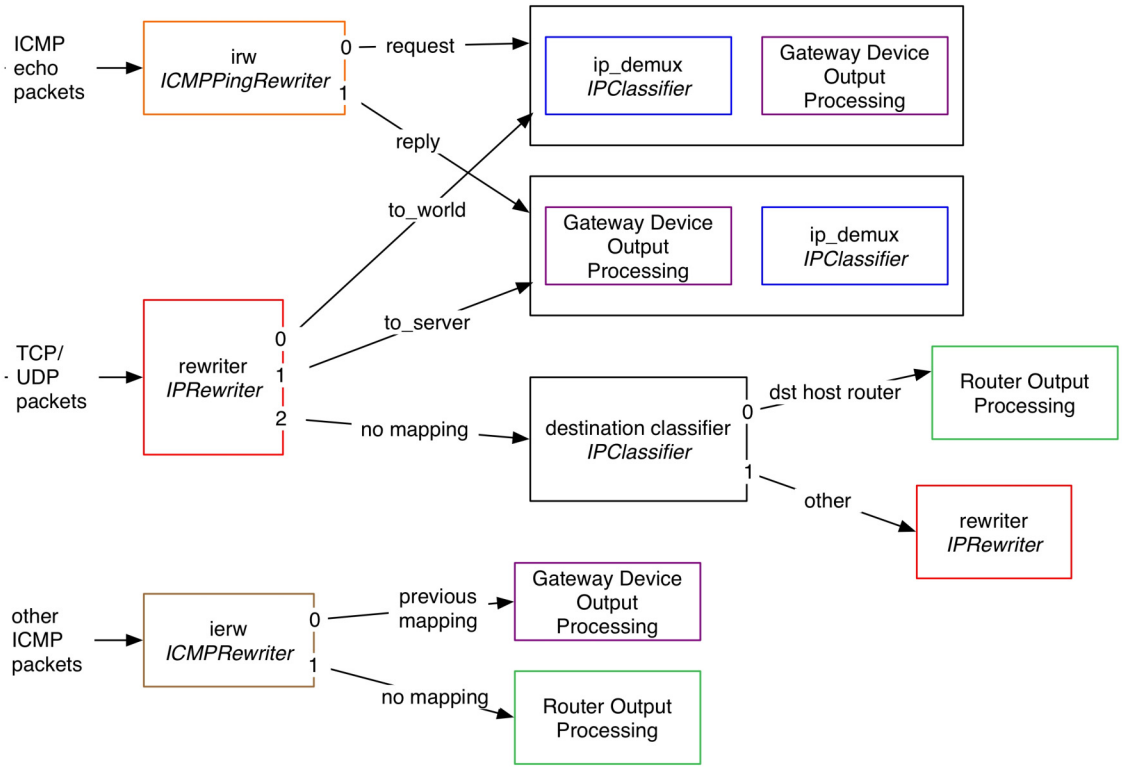


Figure 5-4: Directed Graph Showing the Rewriter Block

In the network implemented, packets passed through the IPRewriter twice. The first pass redirected packets if a reverse mapping existed. Theses packets follow the to_world or to_server edges, depending on the mapping matched. Packets not redirected were classified by destination, and packets destined for the host router were sent to the Router Output Processing block. The remaining packets went back to the IPRewriter, this time to perform NAT. The packets were then sent to the Output Processing block for either the Mesh Wireless Device, or Gateway Device. For example, a TCP packet arriving from the mesh wireless device and destined for the host router would have arrived to the IPRewriter at input 3. If a previous mapping existed, it would be sent to output 1 of the rewriter, otherwise it would be passed to output 2. There it would be forwarded to the Router Output Processing block.

The two passes through the IPRewriter were made to avoid mapping packets destined for the host router, which would cause the router to reply to itself, and the flow to be lost.

The ICMPPingRewriter output all mapped ICMP echo-requests on output zero. This will either lead to the ip_demux IPClassifier, or to the Gateway Device Output Processing, depending on which configuration was used, either the Client Gateway or Internet Gateway. The replies went to output one, which lead to the next block.

The ICMPRewriter simply output packets that matched a mapping on output zero. These were sent to the Client Device Output Processing. Packets without a mapping belonged to the host router, and were sent to the Router Output Processing block.

5.3.4.2. Mesh Wireless Device Input Processing and Packet Classification

This section describes the input processing and packet classification done for packets received from the mesh network. The elements included can be seen in Figure 5-5.

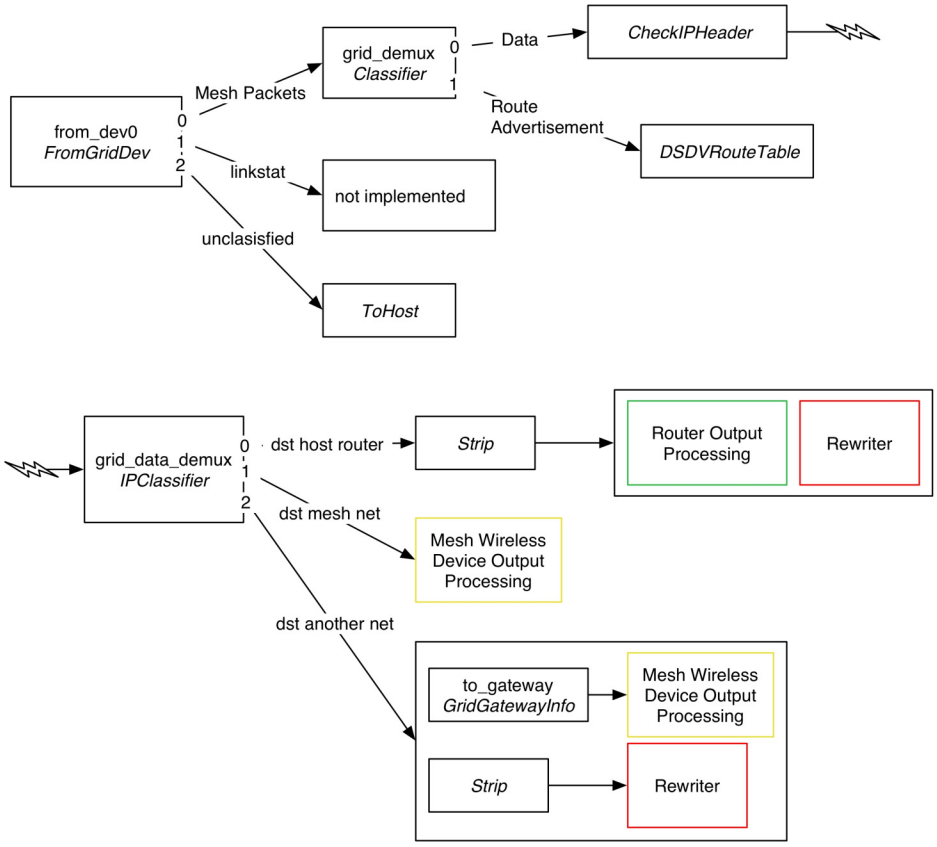


Figure 5-5: Directed Graph of Mesh Wireless Device Input Processing and Packet Classification Block

Packets that were received from the wireless device were classified as mesh data, linkstat data or other packets. If a packet did not originate from another mesh router, it would not have been encapsulated with the proper grid header. It would then have followed the unclassified edge, and be sent directly to the host router. This was useful when debugging the origin of communication problems between two routers. If a packet generated from a system that was not part of the mesh network reached the router, there was a problem with the Input Processing and Packet Classification section.

The router configuration had the ability to determine statistical information about generated "linkstat" packets. This feature, however, was not used, since this project is concerned with end-to-end communication through the mesh. Any other packets, marked as mesh packets, go on to the grid_demux classifier.

These packets were classified as either route advertisements or data. The route advertisement packets were forwarded to the DSDVRouteTable element. The IP header of the data packets were checked for integrity by the CheckIPHeader element, and then the packet was classified according to its destination IP address.

If they were addressed to the current router, the grid header was stripped, and the packets were sent to the Rewriter block if it existed. The Rewriter block does not exist in packet forwarding routers, instead packets are sent to the Router Output Processing block.

If the packets were destined for another router in the mesh network, they were sent to the Mesh device Output Processing block.

If the packets were destined for another network, they were sent on one of the two following paths, depending on the configuration. If the router was a Packet Forwarding or Client Gateway router, the packet was sent to a GridGatewayInfo element. The packets were then sent to the Mesh Wireless Device Output Processing block. If the router was an Internet Gateway router the packets had reached the gateway, and they were sent to the rewriter elements.

5.3.4.3. Mesh Wireless Device Output Processing

Some processing must be done to packets before they were routed through the mesh network using the DSDV protocol. The elements that were implemented to do this are shown in Figure 5-6. Note that colored elements have been duplicated for clarity only.

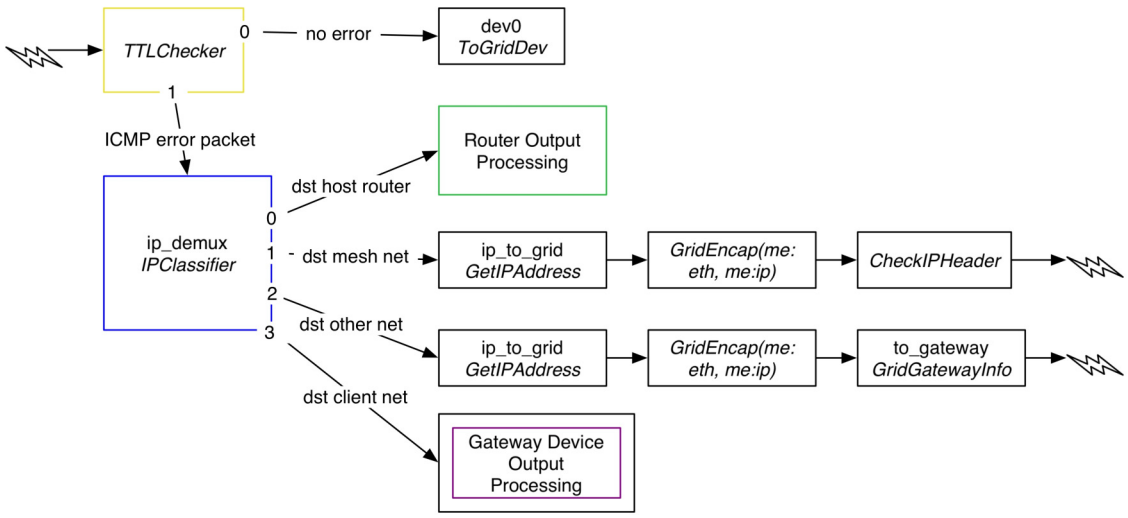


Figure 5-6: Directed Graph of Mesh Wireless Device Output Processing Block

Packets originating from the Mesh Wireless Device Input Processing and Packet Classification block were sent to the TTLChecker element. If the time to live for a packet had not expired the packet was sent to the wireless device. If it had expired, the generated ICMP error packet was sent to the ip_demux IPClassifier. Packets from the Router Input Processing and Packet Classification block also arrived to the output path via the ip_demux element.

Packets arriving at the ip_demux element were classified based on their destination IP address. These destination could be the host router, mesh network, other network, or client network.

If the destination was the host router the packet was sent to the Router Output Processing block.

If the destination was the mesh network, or another network the packets were encapsulated in a grid header containing the IP and MAC addresses of the host router. If these packets were destined for another network, they were sent to the GridGatewayInfo element, to have their destination IP address annotation set to the address of the gateway router. If not, they were sent to a CheckIPHeader element to have their destination IP address annotation set. They were then sent to the TTLChecker element, and handled as described above.

If a packet is addressed to the client adapter's IP address, it is sent to the Router Output Processing block. If it is destined for the client network, it is sent to the Client Device Output Processing block.

5.3.4.4. Router Input Processing and Packet Classification

This section describes the input processing done to packets originating from this router. This would occur if a console session was started at that router or if an administrator was connected

remotely to the router and wanted to connect to another router. The elements included are shown in Figure 5-7.

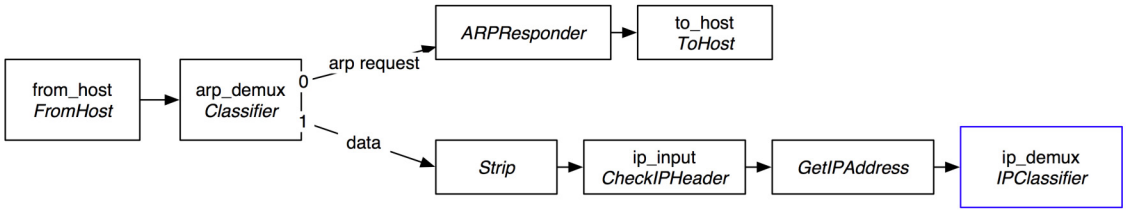


Figure 5-7: Directed Graph of Router Input Processing and Packet Classification Block

The packets from the router were first classified as either ARP request packets, or data. If they were an ARP request, a reply was sent back to the router itself. This was done since the DSDV protocol was used to determine the destination of the packet, not the ARP protocol. This is explained in further detail in the next section.

If the packets were data packets, the Ethernet header was stripped. The IP header was then checked for integrity. If there was an error, the packet was dropped. If not, the destination IP address annotation was set for the packet, and it was sent to the ip_demux IPClassifier.

5.3.4.5. Router Output Processing

This simple block processed packets that were destined for the host router. The elements included in this block are shown in Figure 5-8.

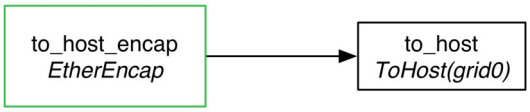


Figure 5-8: Directed Graph of Router Output Processing Block

In the previous section packets classified as ARP requests were sent to the ARPResponder element. This element creates a response to the request using a fake MAC address, so that the Linux kernel would think it had located the machine it was looking for. It would then send out the packets to that machine, so the packets would be sent out with an Ethernet header containing the fake address. This header would be stripped in the Router Input Processing and Packet Classification step.

The EtherEncap element encapsulated the incoming packets in an Ethernet header, using a fake MAC address. This did no harm, since the packets were then sent to the Linux kernel, where this Ethernet header was stripped.

5.3.4.6. Gateway Device Input Processing and Packet Classification

Packets that originated from a client, or the Internet, would have arrived at this block. The ARP protocol was used to match an IP address to a MAC address on the client network. The Internet gateway configuration did not implement this protocol. The elements included in this block can be seen in Figure 5-9.

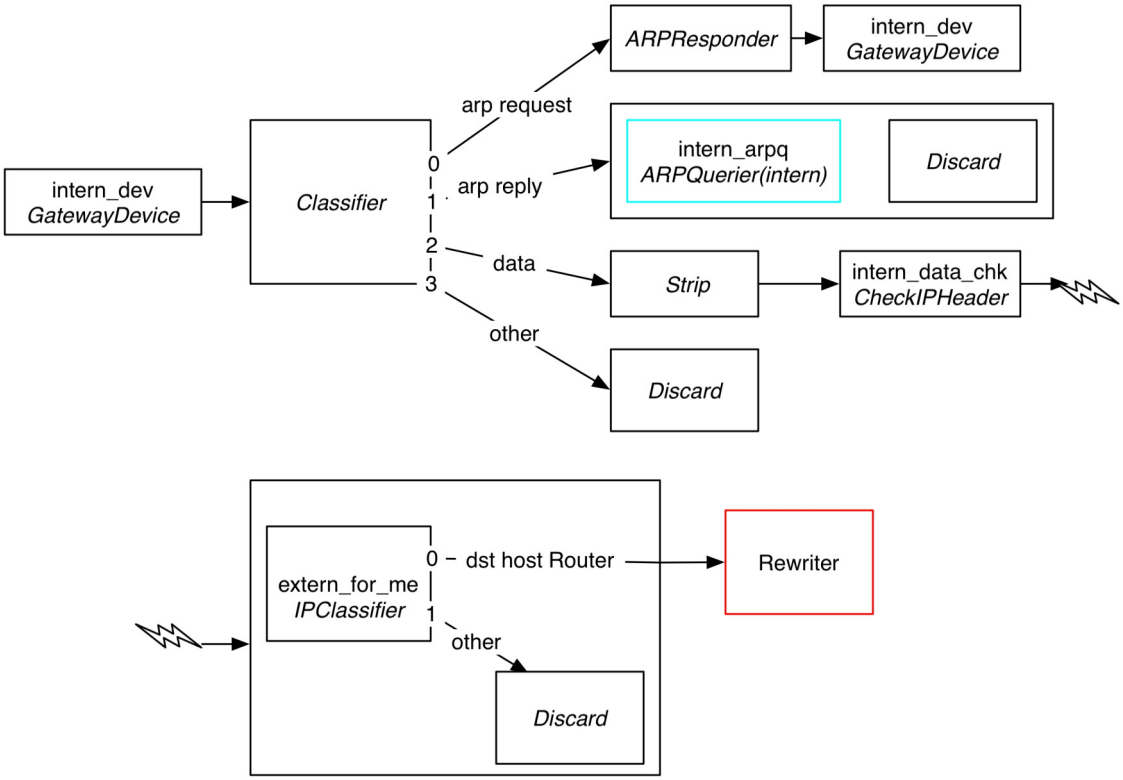


Figure 5-9: Directed Graph of Gateway Device Input Processing and Packet Classification Block

The Classifier first classified all packets. If they corresponded to an ARP request, they were sent to an ARPResponder.

If the incoming packet was an ARP reply packet, it was sent to the first input of the intern_arpq ARPQuerier element. In the Internet Gateway Configuration these packets were discarded. Data packets were sent on for further processing. If a packet could not be classified as either an ARP packet or a data packet, it was discarded.

The data packets had their Ethernet headers stripped, then had the integrity of their IP header checked. In the Internet Gateway Configuration, the packets were then passed to the extern_for_me classifier. This was used to discard all packets that were not addressed to the host

router. Since packets had not gone to a rewriter element at this point, even packets destined for the mesh network would be addressed to the host router. This was not done for the Client Gateway Configuration, since packets would be addressed to a different router, or to a system on an external network.

Packets were then classified by the type of packet using an IPClassifier element. This was done in order to send them to the proper rewriter element in the Rewriter block.

5.3.4.7. Gateway Device Output Processing

This section describes the processing done to packets before they were sent out to the client systems or the Internet. The included elements of this block can be seen in Figure 5-10.

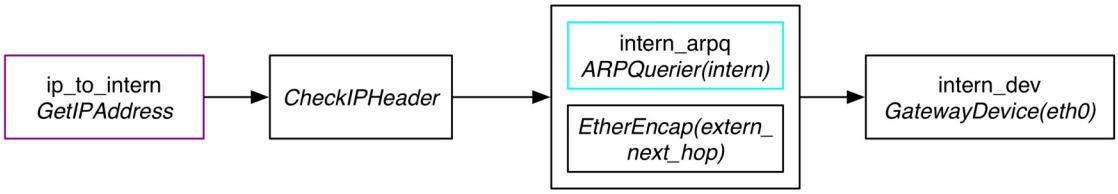


Figure 5-10: Directed Graph of Gateway Device Output Processing

If an ARP request was received on the gateway device, an ARP response was created using the ARPResponder element. The response was addressed to the request sender, and contained the MAC address for the gateway device of the host router. This response was then sent to the gateway device.

Packets arriving at the ip_to_intern element had their destination IP address annotation set and had their IP header integrity checked.

In the Client Gateway Configuration, they were sent to the intern_arpq ARPQuerier element. When the MAC address corresponding to the IP address of the destination was discovered, the packet would be sent to the gateway device.

The Internet Gateway configuration did not use the ARP protocol, since packets were sent to an external router in order to access the Internet. The MAC address of this external router had to be coded in the configuration file prior to setting up the mesh, it was not dynamically discovered. Packets were sent to an EtherEncap element, where they were encapsulated in an Ethernet Header containing the MAC address of the external gateway router. They were then sent to the gateway device.

Chapter 6 – System Automation

When setting up the mesh routers, as with any other consumer router, a monitor and keyboard were not available. Without this requirement, the mesh routers would not have been easily set up in an arbitrary location. For example, during testing, mesh routers were set up along the hallway in the Engineering III building, not in a lab setting. To be able to properly manage the mesh routers, and to confirm that the machine booted properly, a number of utilities and scripts were used. These are presented in the following sections and code is available in Appendix C.

6.1.1 Meshstart

A start up script that used the system beep command to exhibit the mesh router's status, configure the devices, and to start the Click Modular Router software was created. For more information about the Click Modular Router software, see Chapter 5.

There were a number of steps required to setup a system to behave as one of the mesh routers. These included configuring the wireless device, configuring the client interface device if present and needed, and choosing the correct configuration to be used by the Click Modular Router. To make sure that when one of the systems was booted up it had successfully completed all the required steps, a Perl program was written. This program was broken into two separate files, one called meshstart.init the other called meshstart. Meshstart.init was run during the boot up process, and calls meshstart. Meshstart executed the required commands based on a configuration file called meshstart.conf, which defined all of the parameters needed to set up the wireless card for communicating with the other mesh wireless devices. These parameters include the device name, channel to use, network id (ESSID) and the address with which the device should be configured. Also defined in the meshstart.conf file is the client device information. If this configuration information is present, the specified device is configured to allow client machines access to the mesh network through the system. The meshstart.conf file also defines whether or not to run the Click software in userspace or as a kernel module, and what click configuration to use. As the meshstart program runs, and sets up the devices, it beeps in a way to notify if there was a problem with running one of the commands or if everything was successful. A description of the beeps can be found in Table 6-1 below. A failure of a setup step can be heard when after hearing the beep sequence for the step, the failure beep sequence is heard. Meshstart is configured to run during the boot up process, reading the meshstart.conf file and properly configuring the system.

Table 6-1: Mesh Start Beep Sequences

Configuration Step	Corresponding Beep Sequence
Meshstart startup	4 beeps, increasing pitch
Configure mesh wireless device	1 beep
Configure client device, if configured	2 beeps
Start Click	5 beeps, increasing then decreasing pitch
One of the following commands failed	2 beeps, decreasing pitch.

As the network was being debugged, changes to the configuration of the machines were made in the lab and then the machines were deployed for testing. However, once deployed, there was no way of updating the configurations, besides bringing the machine in question back to the lab. To solve this problem, functionality was added to attempt to read the CD-ROM drive of the machine. If at boot up there was a CD-ROM in the CD-ROM drive, the device was mounted for reading, and the meshstart program checked to see if there was a meshstart.conf file on the CD-ROM. If the file was present, meshstart used this configuration file instead of the default one. A CD-ROM could be created with a new configuration, and deployed simply by rebooting the machine in question with the new CD-ROM in the drive. This feature was used multiple times to allow several configurations, such as using the 802.11g standard instead of the 802.11a standard, or using a different configuration file for Click Modular Router to modify the behavior of the mesh router. All source code for these programs can be found in Appendix B

There is one last procedure that meshstart did which helped to alleviate the memory leak problems with the Madwifi driver. Meshstart called another program named memwatch, which is discussed in the next section.

6.1.2 Memwatch

Due to the memory leak issue in the Madwifi driver mesh routers would randomly run out of memory, and crash. It was difficult and time consuming to determine which machine failed, since it became completely unresponsive. As a temporary solution to allow some stability in the network, a Perl program called memwatch was created. The program polled the amount of free memory in the system, and, when it went below 5 Mb, memwatch stopped the Click Modular Router and unloaded the ath_pci kernel module. By doing this, the memory used by it was released. When the ath_pci kernel module was unloaded, the wireless device needed to be

reconfigured, and the Click Modular Router started again. As this is what meshstart did, memwatch then executed meshstart and exited.

The meshstart script redid all the steps previously done during boot up. When meshstart was finished the system was setup once again as a mesh router, and memwatch was started to watch for low memory again.

It was found that during heavy testing memwatch was effective in keeping the mesh routers up and running. However, due to the nature of the mesh routing, by starting and stopping the Click Modular Router, some machines were slow when re-establishing a connection to the mesh network. During internet traffic usage with one or two clients, the memory leak in the Madwifi module was not as noticeable, and memwatch usually did not need to release the memory.

6.1.3 Logging

As there was a need to keep track of the problems that occurred, and to determine the state of the network while the mesh routers were running, relevant data was logged. These logs were especially important during the testing of the implementation, as monitors were not available when the mesh routers were deployed. These logs included the times at which meshstart was run, and any problems that were encountered during its execution. Also logged were the times when memwatch needed to free memory filled by the leak in the module. For general system information, the syslog-ng system logger was used. This program was installed on each machine using Gentoo's Portage distribution system. It logged operating system information such as system initialization data, login times, and all data coming from the kernel. The system logger also kept track of output from the Click Modular Router when it was run as a kernel module. This data allowed the routing table changes and updates to be seen, as well as any errors that the router encountered. Debug packet flow information from the Click Modular Router was also printed to this log.

6.1.4 System Replication

Due to the software requirements of both Click and Madwifi, as well as the manual nature of the Gentoo installation process, the time it took to set up a machine was sizeable. It was also required that each machine be identical to each other. Due to these requirements, a process to copy the entire hard drive from one system onto all the others was created. Due to the nature of the Linux operating system, hard drives are represented as device files. To copy the entire contents of one hard drive to another the commands in Table 6-2 were used.

Table 6-2: Machine Cloning Commands

Machine Cloning Commands
`dd  if=/dev/hda of=/dev/hdb`
`where dev/hda is the hard drive containing the complete system.`
`dev/hdb is the destination hard drive`

Once a host machine was configured with two hard drives, one to be read from and one to be written to, the copy took about 20 minutes. Once the cloned hard drive was installed back into the other machine, there were a few steps required to configure it. The following list summarizes the final steps needed to configure the newly cloned machine:

1. The /etc/conf.d/hostname, /etc/csh.env, /etc/env.d/01hostname, /etc/profile.env files were updated to specify the new hostname.
2. The configuration file for the Click Modular Router was changed to specify the IP address of the new machine.

After these files were updated, a reboot was all that was needed to have the machine configured with all the software requirements of a mesh router.

Chapter 7 - System Integration

This Chapter will describe the entire system as a whole. The first section will describe how individual mesh routers are used to construct the mesh network. The hardware configurations for these devices were described in Chapter 3, and the different router configurations were described in Chapter 5. The second section describes the necessary setup steps for a client, either wired or wireless, to connect to the mesh network.

7.1 Mesh Network Setup

By using the tree configuration for the mesh routers, the mesh network was constructed in a linear topology to maximize the number of hops. The routers were laid out as follows. An Internet gateway router was found at the start of the line. This was connected to the Internet, or to a dedicated server for testing purposes. The test configuration is described in further detail in Chapter 7. After the Internet gateway router there were three packet forwarding routers. At the end of the line was the Client Gateway Router. These routers connected the clients to the mesh network. As will be discussed in the next chapter, by varying the number of Packet forwarding routers between the Internet gateway router and the Client gateway router, the number of hops a packet took changed. This linear topology can be seen in the following figure.

7.2 Client Setup

There are a number of steps that need to be done on a client machine to allow it to connect to the mesh network. Due to the scope of the project, automatic configuration was not implemented, and manual commands needed to be entered. This section describes how a Linux client was connected to the mesh network. The same settings can be specified on a client running a different operating system, such as Microsoft Windows, but the commands to do this would be dependant on the operating system being used.

7.2.1 Wireless Interface Configuration

The wireless device of a client machine needed to be configured with the proper settings to allow it to connect to a Client Gateway Router. The commands that were run on a Linux machine, configured with a wireless network device named eth1, can be found in Table 7-1. The first command sets the mode, channel and network name for the wireless network. The second command sets the proper addressing information for the network interface device. The third

command brings the device up, making it active. The last command sets kernel routing information that designates the address 192.168.1.5 as the gateway. This gateway is where all Internet traffic is sent to, and is the address of the Client Gateway Router. This address must be known in advance. To allow for DNS lookup, the address of a name server must be know in advance and added to the /etc/resolv.conf file.

Table 7-1:Commands to Required to Allow Client Connection to Mesh Network

`iwconfig eth1 mode ad-hoc channel 11 essid clientnet`
`ifconfig eth1 inet 192.168.1.13 netmask 255.255.255.0`
`ifconfig eth1 up`
`route add default gw 192.168.1.5 eth1`

7.2.2 Wired Interface Configuration

The setup for a wired client is mostly identical to the wireless client configuration, without the need for the first command that is specific to wireless devices. For the mesh network created, the address for the wired Client Gateway router was 192.168.1.1. The proper name of the wired Ethernet device was substituted for eth1.

7.3 Complete System

To fulfill the goals of the project, a client must be able to connect to a client gateway router, and through the mesh network access the Internet. This would provide the client the ability to access web sites, send email, and transfer files. With the mesh network setup and configured as described in Section 6.1, and having a client configured as described in Section 6.2, this was achieved. The ability to connect to the Internet, through the mesh network utilizing all the mesh routers available in this project was successful. The machines were setup along the hallways in the Engineering III building, with the distances between machines great enough to force the maximum number of hops. If the distance between two machines was too small, then there was a possibility of an intermediate system being skipped. This situation was avoided by checking the routing table on each mesh router. See [Appendix B] for an example of the routing tables observed. A group member's laptop was configured as a client, and was able to send email, browse websites, and transfer files from the Internet. There was a noticeable difference in performance when accessing the Internet over four hops as opposed to one hop, which was achieved in the laboratory. A figure representing the complete system can be seen in Figure 7-1, note that to simplify the figure two of the packet forwarding routers are not shown.

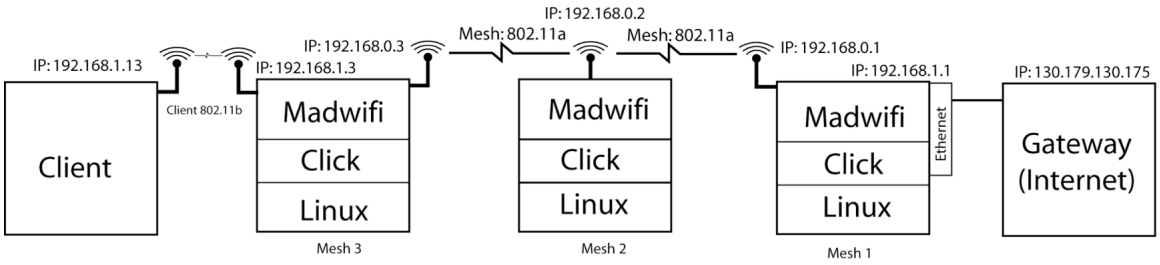


Figure 7-1: Block Diagram of the Complete Wireless Mesh Network

To gain insight into the performance of a wireless mesh network, there were a number of tests planned, including varying the number hops, and examining the fairness between clients at different location within the mesh network. Throughput and round trip time were to be used to measure these tests. To make the measurements meaningful, the network would have to be stressed with high amount of traffic. When beginning to test the mesh network as described, the mesh routers were starting to crash. As mentioned in previous sections, the cause of the crashes was attributed to a memory leak in the Madwifi driver.

Figure 7-2: Photo of Complete Wireless Mesh Systems

Chapter 8 - Testing Methodology and Analysis

The chapter reviews the three different testing methodologies used and the methods of analysis and results of analysis. Figure 8-1shows a general linear testing topology.

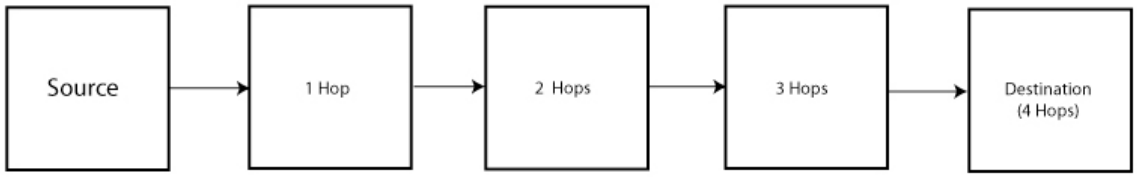


Figure 8-1: Basic Linear Topology

This was selected to maximize the number of hops with the available hardware, and to reduce testing complexity. It was also a standard configuration, allowing the results obtained to be applied to more general cases. Within the linear topology two testing metrics were used to evaluate system performance: throughput and packet delay (RTT). To test these metrics efficiently three different tests are required: The effect of varying the number of hops, the effect of a heavily loaded system, and the fairness with respect to clients connected at different mesh routers. The tests selected were based on the goals set out for the project to examine standard internet traffic. This as mentioned in Chapter 6 were performed in the engineering 3 hallway, spaced out to ensure not that there was no overlapping signals from the mesh router. The testing was adversely affected by the memory leak problem described in Chapter 6, and test data is understandably incomplete.

8.1 Preliminary Internet Test

A five hop network was established, data in figure shows the connectivity of the network (see Appendix D for test data). Standard web browsing, file downloading and email were performed quickly and efficiently by a client. This was done using both the user and kernel level of the system. Using a web based bandwidth tests it was found that the running the Click Modular router as a kernel module was noticeably faster than using the user level driver. Since the difference was so clear, testing was done by running the click modular router as a kernel module in order to obtain the maximum response out of the network.

8.2 Effect of Number of Hops on the Network

A major reduction in efficiency in a wireless mesh network is cause by the number of hops data has to travel through to reach its final destination. The test examined the effect of varying the

number of hops on the Throughput and average packet delay (RTT) in both the TCP and UDP protocols on the network. To test TCP traffic though the network a basic file transfer of a 5MB file was used. To test the effect of UDP packets on the system, a continuous UDP packet stream at a data rate of 200packets/s was applied to the system for 30 seconds using the D-ITG traffic generator.

Both tests are to represent common applications of networks and the internet. The TCP test represents a common file download where the UDP test represents a rudimentary example of a streaming application such as video or audio. Each of the tests was performed for one through three hops and repeated five times at each interval. Throughput and RTT were examined for each test and recorded. As seen in Table 8-1 the testing parameters for each test are outlined.

Table 8-1: Testing Parameters for Number of Hops

	Number of Hops	
	TCP	UDP
Runtime	Time to complete	30s
# of Tests	5	5
Metrics	Throughput	Throughput
	Avg. Pkt. Delay	Avg. Pkt. Delay
Number of Hops	3	3
Data Recorder	Ethereal	D-ITG
Notes	Conection made through SSH File transfer with SSH Server and Client	D-ITG Test used with 200 pkts/sec running

In these tests we ran into a number of problems, as mentioned before a memory leak in the madwifi driver, the tests had to be repeated often most of the time finishing incomplete due to a system needing to unload and reload both Click and the driver to prevent a system crash. By the third and fourth hop it was very difficult to complete five tests since usually at least one system would crash during a test.

The data acquired from these tests are shown in Table 8-2 below.

Table 8-2: Averaged Collected Data from No. of Hops Tests

Test Type/ Number of Hops	1 Hop	2 Hops	3 Hops
TCP: Test 5MB File Download			
Avg. Thoughput (Bps)	201723	213967	159735
Avg. Round Trip Time (s)	0.00054	0.00046	0.00034
UDP: Date Stream Test			
Avg. Thoughput (Bps)	102169	41920	92365
Avg. Round Trip Time (s)	0.01294	0.09978	0.01458

8.3 System Load

This test was used to examine how well the network handled a high load. This was examined at 5 hops and the number of packets was adjusted per second until the network suffered from a packet loss rate of 10% or more. D-ITG was used to test both TCP and UDP protocols on the system and data recorded.

Acquisition of data for high load situations was not feasible due to the memory leak in the Madwifi driver. System memory was used up very quickly causing the memwatch script to reload both Click and the driver. Although the memwatch program was in place to add stability to the mesh network, whenever a mesh router was restarted the traffic flow stopped, and all packets were dropped until the mesh router came back online and was recognized by its neighbors. This meant that the load the network could handle was arbitrarily small, since any amount of traffic would eventually cause a mesh router to temporarily leave the network. This intermittent failure of nodes throughout the system, especially when testing a linear topology caused any collected data to not be representative of network performance and the data was dismissed.

8.4 Fairness

The definition of fairness in a network is equitable treatment of separate clients on a mesh network sharing the same physical medium and channels. In multi hop networks throughput is reduced as you increase the number of hops.[9] Fairness will be examined with the last statement in mind to test effect this has in terms of throughput and packet delay to the clients.

The testing was performed setting a client at the final hop and moving the second client through all other hops and examining the throughput of the network on each client though ethereal. Figure 8-2 shows the topology used for testing where client one fixed and the other clients represented by the dotted lines move through the network.

This test again ran into major problems with the memory leak due to nodes randomly failing, not allowing for any consistent results and causing inaccurate data.

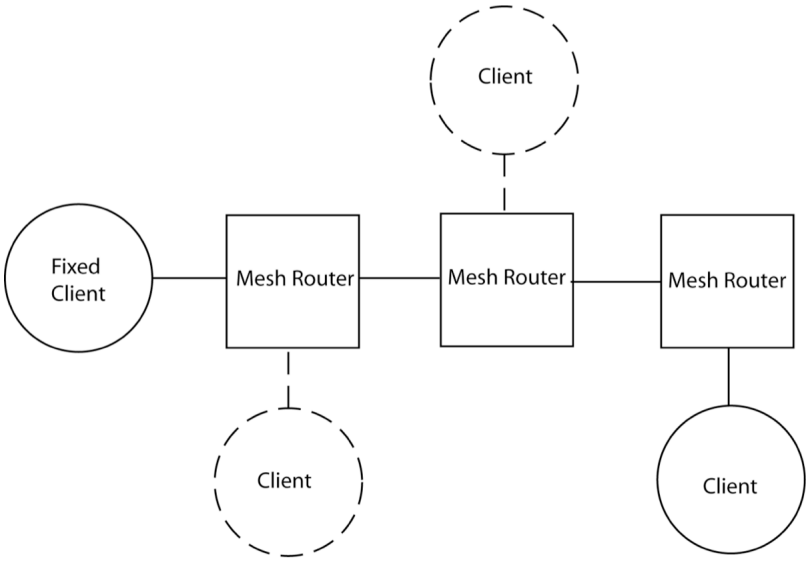


Figure 8-2: Fairness Test Topology

8.5 Results and Analysis

The section will outline and analyze the results acquired from testing. Unfortunately only partial data will be presented due to the memory leak in the Madwifi driver already mentioned. The DSDVRouteTable data in Appendix D shows an Internet connection through five hops could be competed and the network was successful. Data collected from the Number of hops test looking to Table 8-2 were inconclusive. For the TCP test, throughput did reduce significantly between hop 1 and hop 3 but where hop 2 increased slightly. In contrary, the RTT decreased as the number of hops increased which was not an expected result. This can be attributed again to the lack of stability in the network due to memory leak in the Madwifi driver. The UDP test showed very little correlation between any of the data points. In addition to the previously mentioned problem, these packets were unguided and did not have any control protocol to prevent dropped packets or initiate recovery, which may have attributed to the inaccuracies.

From the tests above it was shown that a network connection was successful under low load situations for short periods of time. Network instability was the major factor during testing load situations causing data to be inaccurate.

Chapter 9 - Future Work and Conclusion

In this design project a wireless mesh network using off the shelf components was developed. The first goal of the project was to design and implement a wireless mesh network to support standard user Internet traffic. Also, the network should be easily deployable and reduce infrastructure costs. The last goal would be to test the effect of the number of hops and the load produced by the number of clients on the performance of the designed network.

A working wireless mesh network was created using standard off the shelf computer hardware. This allowed a user at a client machine access to the Internet through the mesh network. Common tasks such as email, web browsing and file transfers were possible through four hops, the maximum allowed by the hardware resources of this project. This report describes the software systems needed to support the hardware used in this project. This software was also needed as the underlying layer for the Click Modular Router software router. These systems interacted together to form a functional wireless mesh network that could be deployed with little existing infrastructure. As such, the first three goals of the project have been satisfied.

However, the network was not stable under high traffic loads, due to a memory leak in the Madwifi driver used for the D-Link DWL-AG530 wireless card used for mesh communication. This affected the testing of the designed network, limiting the data that could be acquired.

The design demonstrates how a flexible networking scheme can be designed with consumer hardware. Stable support for the hardware configured in the manner presented is not yet available. Due to the nature of the open source driver, the source code is constantly evolving and a solution should be available soon. Once the memory management problem described is solved, much future work could be done to build on this project.

Once performance data is available for the mesh network as outlined, different routing protocols could be explored. Since the Click Modular Router was used, different available protocols could be integrated into the existing configuration files easily.

Different topologies could also be explored. Another interesting topology is the grid topology, allowing multiple routes from a source to a destination. The effect of routing based on a combination of metrics such as the number of hops, signal strength or packet delay could also be explored.

The network also lacked many usability features, which where out of the scope of this project. The routers could be implemented on smaller hardware, reducing the footprint of the system. Wireless network security could be added. Dynamic configuration of the clients could be

implemented by running a DHCP server on the client gateway router. Routers could acquire network information so hard coded values in the configuration files would not be needed.

Chapter 10 - References

[1] Hill Associates Inc., Telecommunications: A Beginner's Guide, Berkley, Ca.: McGraw-Hill/Osborne, 2002.

[2] R. L. Freeman, Fundamentals of Telecommunications, 2nd ed. , Hoboken, N.J.: John Wiley & Sons Inc., 2005.

[3] I. F. Akyildiz, "A survey on wireless mesh networks," In IEEE Radio Communications, Sept., pp. 23-30, 2005

[4] C. E. Perkins and P. Bhagwat, "Highly dynamic Destination-Sequenced Distance-Vector routing (DSDV) for mobile computers," In Proc. ACM SIGCOMM Conference '94), August 1993, pp. 234–244.

[5] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek. "The Click modular router," ACM Transactions on Computer Systems 18(3), August 2000, pp. 263-297.

[6] S. Avallone, A. Pescapè and G. Ventre, " Distributed Internet Traffic Generator (D -ITG): analysis and experimentation over heterogeneous networks," in ICNP 2003 poster Proceedings, International Conference on Network Protocols 2003, November 2003

[7] Y. Lu, W. Wang, Y. Zhong, B. Bhargava, "Study of Distance Vector Routing Protocols for Mobile Ad Hoc Networks," In Proc. IEEE PerCom '03, 2003, pp. 97 – 114.

[8] J. Broch, D. A. Maltz, D. B. Johnson, Y. Hu, and J. Jetcheva. "A performance comparison of multi-hop wireless ad hoc network routing protocols," In Proc. ACM/IEEE MobiCom 1998, 1998, pp. 85–97.

[9] T. Kuang, and C. Williamson, "A Bidirectional Multi-channel MAC Protocol for improving TCP Performance on Multihop wireless Ad-hoc Networks," in 7th ACM International Symposium on Modeling, analysis, and simulation of wireless and mobile sytems, 2004, pp.301-310

Appendix A - Click Elements

Classifier

This element, along with the IPClassifier, formed the basis of how packets flowed through the click router. It classified all the input packets based on an offset, and a bit value. Each of these patterns had an associated output. If a packet matches the bit value at the given offset, they were sent out the corresponding output. A special pattern, '-', matched every packet, and could be used as the last pattern in order to route all unmatched packets. This element formed a construct similar to many programming languages switch/case statement.

IPClassifier

This element functioned in the same manner as the Classifier, except that it classified packets based on their type, or the protocol they used. For example, it could distinguish echo-request ICMP packets from ICMP error packets. It could also classify packets based on their destination or source IP addresses.

ARPQuerier

This element was used to discover system MAC addresses using the ARP protocol. It contained a table of MAC addresses associated with IP addresses. If a packet arrives on input 0, and there is no mapping, an ARP request is sent out. When the ARP reply arrived on input 1, the table was updated and the packet was sent out.

GetIPAddress

This element sets the destination IP address annotation.

CheckIPHeader

This element checks the integrity of the IP header of a packet. It also sets the destination IP address annotation for that packet.

TTLChecker - Compound Element

This compound element was used to decrement the time to live (TTL) for a packet if it was being forwarded to another router. Classifiers were used to decrement only the TTL for IP packets that did not originated from the host router. All other packets were just sent to the output. The TTL is decremented using the DecIPTTL element. If it had not expired, the checksum for the packet was

recalculated, and the packet was sent out on output 0. If it had expired, the packet was sent to an ICMPError packet generator element. This generated an ICMP Error packet with the error code ‘timeexceeded’ (11). This packet was addressed to the original packet sender, and was sent out on output 1 of the TTLChecker compound element.

Appendix B - Click Modular Router Configuration Files

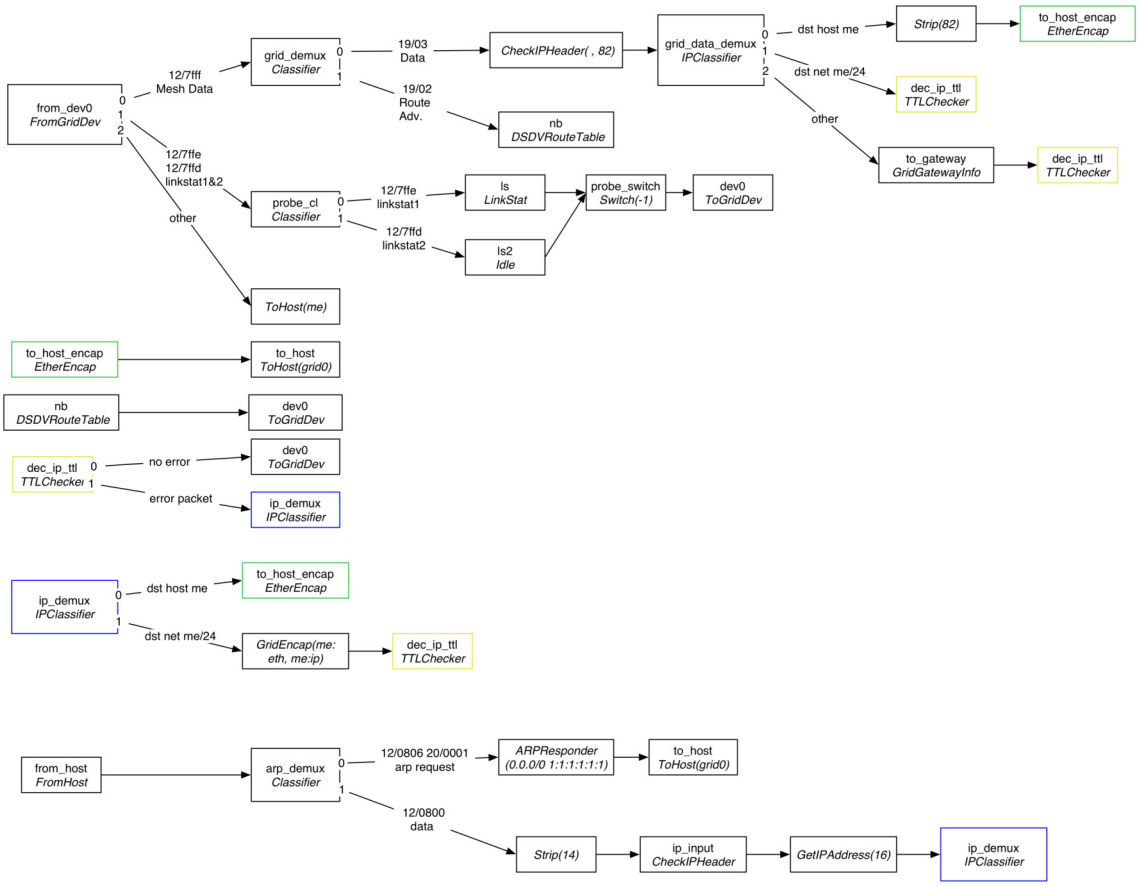
dsdv_kernel.click

```
// This file automatically generated at Thu Feb  2 10:57:28 CST 2006 with the following
command:
// make-dsdv-config.pl -k -i ath0 -a 192.168.0.2

// this configuration performs routing lookup *after* the interface
// queue, and only works with one interface.

AddressInfo(me 192.168.0.2 ath0:eth);

elementclass TTLChecker {
  // expects grid packets with MAC headers --- place on output path to
  // decrement the IP TTL for next hop and provide traceroute support.
  //
  // push -> push
  //
  // output [0] passes through the Grid MAC packets
  //
  // output [1] produces ICMP error packets to be passed back to IP
  // routing layer

  input -> cl :: Classifier(19/03, -);
  cl [1] -> output; // don't try to dec ttl for non-IP packets...

  cl [0]
    -> MarkIPHeader(82)
    -> cl2 :: IPClassifier(src host != me, -);

  cl2 [0]-> dec :: DecIPTTL; // only decrement ttl for packets we don't originate
  cl2 [1] -> output;

  dec [0] -> output;
  dec [1] -> ICMPError(me, 11, 0) -> [1] output;
};

li :: GridLocationInfo2(0, 0, LOC_GOOD false);

elementclass FixupGridHeaders {
  $li | // LocationInfo element
  input
    -> FixSrcLoc($li)
    -> SetGridChecksum
    -> output;
```

```
};

elementclass ToGridDev {
  // push, no output
  $dev |
  input -> cl :: Classifier(12/7ffe, // LinkStat 1
                            12/7ffd, // LinkStat 2
                           19/02,
                           19/03);
  prio :: PrioSched;
  cl [0] -> probe_counter :: Counter -> probe_q :: Queue(5) -> [0] prio;
  cl [1] -> probe_counter;
  cl [2] -> route_counter :: Counter -> route_q :: Queue(5) -> FixupGridHeaders(li) ->
[1] prio;
  cl [3] ->  data_counter :: Counter ->  data_q :: Queue(5)
    -> data_counter_out :: Counter
    -> tr :: TimeRange
    -> lr :: LookupLocalGridRoute2(me:eth, me:ip, nb)
    -> Print(lr->)
    -> FixupGridHeaders(li)
    -> data_counter_out2 :: Counter
    -> tr2 :: TimeRange
    -> [2] prio;
  prio
    -> dev_counter :: Counter
    -> t :: PullTee
    -> ToDevice($dev);
  t [1] -> SetTimestamp -> ToHostSniffers($dev);
};

elementclass FromGridDev {
  // push, no input
  // `Grid' packets on first output
  // `LinkStat' packets on second output
  $dev, $mac |
  FromDevice($dev, PROMISC false)
    -> t :: Tee
    -> HostEtherFilter($mac, DROP_OWN true)
    -> cl :: Classifier(12/7fff, 12/7ffe, 12/7ffd, -);
  cl [0]  // `Grid' packets
    -> ck :: CheckGridHeader
    -> [0] output;
  cl [1]  // `LinkStat 1' packets
    -> [1] output;
  cl [2]  // `LinkStat 2' packets
    -> [1] output;
  cl [3] // everything else
    -> [2] output;
  t [1] -> ToHostSniffers($dev);
```

```
  ck [1] -> Print('Bad Grid header received', TIMESTAMP true, NBYTES 166) -> Discard;
};

elementclass GridLoad {
  // push, no input

  // DATASIZE should be the size of the desired UDP packet (including
  // ethernet, Grid, and IP headers), plus 2 for alignment.  It must
  // be at least 120.  Most of this is stripped off to be re-used
  // later, avoiding expensive pushes in the UDP/IP and Grid
  // encapsulation.
  src :: InfiniteSource(ACTIVE false, DATASIZE 120)
    -> Strip(112) // 14 + 60 + 8 + 20 + 8 + 2 = 112
                  // (eth + grid + grid_encap + ip + udp + 2 for alignment)
    -> seq :: IncrementSeqNo(FIRST 0, OFFSET 0)
    -> SetIPAddress(me)
    -> StoreIPAddress(4)
    -> udp :: UDPIPEncap(me, 1111, 0.0.0.0, 8021)
    -> count :: Counter
    -> tr :: TimeRange
    -> output;

  ph :: PokeHandlers;
}

ls2 :: Idle;
ls :: LinkStat(ETH me:eth, SIZE 148 );
metric :: HopcountMetric();

//this is not a gateway
to_gateway :: GridGatewayInfo(nb, false);
nb :: DSDVRouteTable(60000, 15000, 7500, 1000,
                     me:eth, me:ip,
                     MAX_HOPS 100,
                      METRIC metric,
                     VERBOSE false,
                     GW to_gateway

                      );

grid_demux :: Classifier(19/03,    // encapsulated (data) packets
                         19/02);   // route advertisement packets

arp_demux :: Classifier(12/0806 20/0001, // arp queries
                        12/0800);        // IP packets

// handles IP packets with no extra encapsulation
ip_demux :: IPClassifier(dst host me,    // ip for us
                         dst net me/24); // ip for Grid network
```

```
// handles IP packets with Grid data encapsulation
grid_data_demux :: IPClassifier(dst host me,    // ip for us
                                dst net me/24, // ip for Grid network
                                - );           //other packets

// dev0
dev0 :: ToGridDev(ath0);
from_dev0 :: FromGridDev(ath0, me:eth)
from_dev0 [0] -> Paint(0) -> grid_demux
from_dev0 [1] -> Paint(0) -> probe_cl :: Classifier(12/7ffe, 12/7ffd);

probe_cl [0] -> ls ->  probe_switch :: Switch(-1) -> dev0;
probe_cl [1] -> ls2 -> probe_switch;

// support for traceroute
dec_ip_ttl :: TTLChecker -> dev0;
dec_ip_ttl [1] -> ip_demux;

//tee the data, strip the grid header, print the packet
//if no print is required, use this line:
//grid_demux [0] -> CheckIPHeader( , 82)-> grid_data_demux;
grid_demux [0] -> CheckIPHeader( , 82)->print_tee :: Tee;
print_tee[0] -> grid_data_demux;
print_tee[1] -> Strip(82) -> IPPrint(grid_demux[0]->) -> Discard;

grid_demux [1] -> nb -> dev0;

ip_input :: CheckIPHeader -> GetIPAddress(16) -> ip_demux;

to_host :: ToHost(grid0);
to_host_encap :: EtherEncap(0x0800, 1:1:1:1:1:1, 2:2:2:2:2:2) -> to_host;
from_host :: FromHost(grid0, me/24) -> arp_demux -> ARPResponder(0.0.0.0/0 1:1:1:1:1:1) -
> to_host;
arp_demux [1] -> Strip(14) -> ip_input;
from_dev0 [2] -> ToHost(me);

ip_demux [0] -> to_host_encap;  // loopback packet sent by us, required on BSD userlevel
ip_demux [1] -> GridEncap(me:eth, me:ip) -> dec_ip_ttl;   // forward packet sent by us

grid_data_demux [0] -> Strip(82) -> to_host_encap;  // receive packet from net for us
grid_data_demux [1] -> dec_ip_ttl;                                  // forward packet from
net for someone else
grid_data_demux [2] -> to_gateway -> dec_ip_ttl;    //forward packets destined for the
outside world to the gateway
```


from_dev0
FromGridDev
0
1
2
12/7fff
Mesh Data
12/7ffe
12/7ffd
linkstat1&2
other
grid_demux
Classifier
0
1
19/03
Data
19/02
Route
Adv.
CheckIPHeader(, 82)
nb
DSDVRouteTable
grid_data_demux
IPClassifier
0
1
2
dst host me
dst net me/24
other
Strip(82)
to_host_encap
EtherEncap
dec_ip_ttl
TTLChecker
to_gateway
GridGatewayInfo
dec_ip_ttl
TTLChecker
probe_cl
Classifier
0
1
12/7ffe
linkstat1
12/7ffd
linkstat2
ls
LinkStat
ls2
Idle
probe_switch
Switch(-1)
dev0
ToGridDev
ToHost(me)
to_host_encap
EtherEncap
to_host
ToHost(grid0)
nb
DSDVRouteTable
dev0
ToGridDev
dec_ip_ttl
TTLChecker
0
1
no error
error packet
dev0
ToGridDev
ip_demux
IPClassifier
ip_demux
IPClassifier
0
1
dst host me
dst net me/24
to_host_encap
EtherEncap
GridEncap(me:
eth, me:ip)
dec_ip_ttl
TTLChecker
from_host
FromHost
arp_demux
Classifier
0
1
12/0806 20/0001
arp request
12/0800
data
ARPResponder
(0.0.0/0 1:1:1:1:1:1)
to_host
ToHost(grid0)
Strip(14)
ip_input
CheckIPHeader
GetIPAddress(16)
ip_demux
IPClassifier

dsdv_nat_kernel.click

```
// This file is based on the automatically generated file at Thu Dec 29 18:19:01 CST 2005
with the following command:
// make-dsdv-config.pl -i ath0 -a 192.168.0.3 -u

// the DSDV portion of this configuration performs routing lookup *after* the interface
// queue

//The NAT is based off of the mazu-nat and thomer-nat examples available in the
click/conf/ directory

//One device (ath0) will communicate with the grid mesh network using the DSDV protocol,
while the other
//device (eth0) will communicate with an  internal network. Traffic will be forwarded to
and from a server
// with IP 192.168.1.13


AddressInfo(
//me is the mesh network address of this router
me 192.168.0.5 00:11:95:F8:AA:B9,
//intern is the address of the non-grid device
intern 192.168.1.5 00:90:D1:08:5E:7A,
//intern_server is the ip address to which incoming packets are re-directed
intern_server  192.168.1.13

);

//Internal network (non-mesh) gateway device
elementclass GatewayDevice {
  $device |
  from :: FromDevice($device)
        -> output;
  input -> q :: Queue(1024)
        -> to :: ToDevice($device);
  ScheduleInfo(from .1, to 1);
}



elementclass TTLChecker {
  // expects grid packets with MAC headers --- place on output path to
  // decrement the IP TTL for next hop and provide traceroute support.
  //
  // push -> push
```

```
  //
  // output [0] passes through the Grid MAC packets
  //
  // output [1] produces ICMP error packets to be passed back to IP
  // routing layer

  input -> cl :: Classifier(19/03, -);
  cl [1] -> output; // don't try to dec ttl for non-IP packets...

  cl [0]
    -> MarkIPHeader(82)
    -> cl2 :: IPClassifier(src host != me, -);

  cl2 [0]-> dec :: DecIPTTL; // only decrement ttl for packets we don't originate
  cl2 [1] -> output;

  dec [0] -> output;
  dec [1] -> ICMPError(me, 11, 0) -> [1] output;
};

li :: GridLocationInfo2(0, 0, LOC_GOOD false);

elementclass FixupGridHeaders {
  $li | // LocationInfo element
  input
    -> FixSrcLoc($li)
    -> SetGridChecksum
    -> output;
};

elementclass ToGridDev {
  // push, no output
  $dev |
  input -> cl :: Classifier(12/7ffe, // LinkStat 1
                            12/7ffd, // LinkStat 2
                           19/02,
                           19/03);
  prio :: PrioSched;
  cl [0] -> probe_counter :: Counter -> probe_q :: Queue(5) -> [0] prio;
  cl [1] -> probe_counter;
  cl [2] -> route_counter :: Counter -> route_q :: Queue(5) -> FixupGridHeaders(li) ->
[1] prio;
  cl [3] ->  data_counter :: Counter ->  data_q :: Queue(5)
    -> data_counter_out :: Counter
    -> tr :: TimeRange
    -> lr :: LookupLocalGridRoute2(me:eth, me:ip, nb)
    -> FixupGridHeaders(li)
    -> data_counter_out2 :: Counter
    -> tr2 :: TimeRange
```

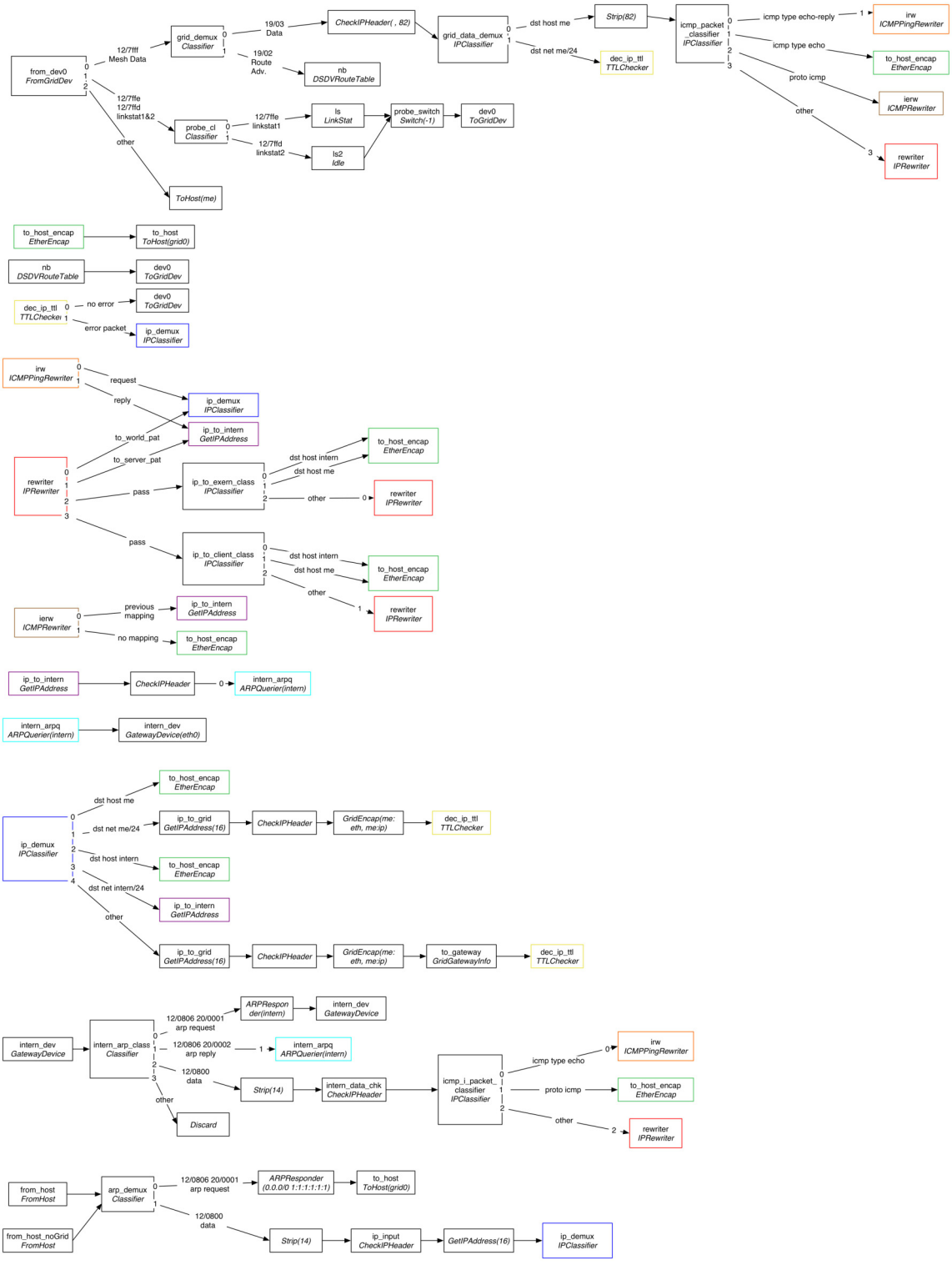
```
    -> [2] prio;
  prio
    -> dev_counter :: Counter
    -> t :: PullTee
    -> ToDevice($dev);
  t [1] -> SetTimestamp -> Discard;
};

elementclass FromGridDev {
  // push, no input
  // `Grid' packets on first output
  // `LinkStat' packets on second output
  $dev, $mac |
  FromDevice($dev, PROMISC false)
    -> t :: Tee
    -> HostEtherFilter($mac, DROP_OWN true)
    -> cl :: Classifier(12/7fff, 12/7ffe, 12/7ffd, -);
  cl [0]  // `Grid' packets
    -> ck :: CheckGridHeader
    -> [0] output;
  cl [1]  // `LinkStat 1' packets
    -> [1] output;
  cl [2]  // `LinkStat 2' packets
    -> [1] output;
  cl [3] // everything else
    -> [2] output;
  t [1] -> Discard;
  ck [1] -> Print('Bad Grid header received', TIMESTAMP true, NBYTES 166) -> Discard;
};

// handles IP packets with no extra encapsulation
ip_demux :: IPClassifier(dst host me,    // ip for us
                         dst net me/24,  // ip for Grid network
                         dst host intern, //ip for us
                         dst net intern/24, //ip for internal net
                         -);            // rest to gateway

//demux for the FromHost processing
arp_demux :: Classifier(12/0806 20/0001, // arp queries
                        12/0800);        // IP packets

//packets input from the router
ip_input :: CheckIPHeader -> GetIPAddress(16) -> ip_demux;

// packets to or from the router
//to_host_encap :: KernelTun(me/24, HEADROOM 68, MTU 1432) -> ip_input;
to_host ::  ToHost(grid0);
to_host_encap :: EtherEncap(0x0800, 1:1:1:1:1:1, 2:2:2:2:2:2) -> to_host;
```

```
from_host :: FromHost(grid0, me/24) ->arp_demux ->ARPResponder(0.0.0.0/0 1:1:1:1:1:1)-
>to_host;
//from_host for the non-grid device
from_host_noGrid :: FromHost(intern0, intern/24) ->arp_demux;

//non-grid device
intern_dev :: GatewayDevice(eth0);


// ARP MACHINERY for non-grid device
intern_arp_class :: Classifier(12/0806 20/0001, 12/0806 20/0002, 12/0800, -);
intern_arpq :: ARPQuerier(intern);

intern_dev -> intern_arp_class;
intern_arp_class[0] -> ARPResponder(intern)   //respond to ARP queries
       -> intern_dev;

intern_arp_class[1] ->  [1]intern_arpq;

//packets from output 2 are data packets (non-ARP). They are dealt with below.

intern_arp_class[3] -> Discard;              //discard unrecognized packets


// REWRITERS for NAT

IPRewriterPatterns(to_world_pat me 50000-65535 - -,
              to_server_pat intern 50000-65535 - -);

rewriter :: IPRewriter(// internal traffic to outside world
                pattern to_world_pat 0 1,
                // external traffic from mesh grid
                pattern to_server_pat 1 0,
                //pass through packets from client
                //to check for previous routes
                pass 2,
                //pass through packets from mesh grid
                //to check for previous routes
                pass 3
);


// Rewriting rules for ICMP packets
irw :: ICMPPingRewriter(me, -);

// Rewriting rules for ICMP error packets
ierw :: ICMPRewriter(rewriter irw);

//output path for packets destined for the internal network
```

```
ip_to_intern :: GetIPAddress(16)
      -> CheckIPHeader
      -> [0]intern_arpq
      -> intern_dev;

// to grid mesh network from internal network
rewriter[2]->ip_to_extern_class :: IPClassifier(dst host intern,dst host me, -);
   ip_to_extern_class[0]->to_host_encap;    //packets addressed to the router
   ip_to_extern_class[1]->to_host_encap;
   ip_to_extern_class[2]-> [0]rewriter;    //packets addressed to another machine go to
the grid mesh network

   rewriter[0]->ip_demux;

// to internal server from grid mesh network

rewriter[3]->ip_to_client_class :: IPClassifier(dst host intern,dst host me, -);
   ip_to_client_class[0]->to_host_encap;    //packets addressed to the router
   ip_to_client_class[1]->to_host_encap;
   ip_to_client_class[2]-> [1]rewriter;    //packets addressed to another machine go to
the grid mesh network

rewriter[1] -> ip_to_intern;

//ICMP Ping packets:
irw[0] -> ip_demux;
irw[1] -> ip_to_intern;

//OTHER ICMP packets
ierw[0] -> ip_to_intern;
ierw[1] -> to_host_encap;  //packets that were not re-written
// FILTER & REWRITE IP PACKETS FROM INSIDE

intern_arp_class[2] -> Strip(14)    //data packets
       -> intern_data_chk :: CheckIPHeader
       -> icmp_i_packet_classifier :: IPClassifier (icmp type echo, proto icmp, -);

       icmp_i_packet_classifier[0] -> [0]irw;     // icmp ping
       icmp_i_packet_classifier[1] -> to_host_encap;        // other icmp
       icmp_i_packet_classifier[2] -> [2]rewriter;  // stuff for the mesh network or for
this router



// Grid mesh network packets

ls2 :: Idle;
ls :: LinkStat(ETH me:eth, SIZE 148 );
metric :: HopcountMetric();
```

```
// don't advertise as a gateway
to_gateway :: GridGatewayInfo (nb, false);
nb :: DSDVRouteTable(60000, 15000, 7500, 1000,
                     me:eth, me:ip,
                     MAX_HOPS 100,
                      METRIC metric,
                     GW to_gateway,
                     VERBOSE true

                      );

grid_demux :: Classifier(19/03,    // encapsulated (data) packets
                         19/02);   // route advertisement packets


// handles IP packets with Grid data encapsulation
grid_data_demux :: IPClassifier(dst host me,    // ip for us
                                dst net me/24); // ip for Grid network

// dev0
dev0 :: ToGridDev(ath0);
from_dev0 :: FromGridDev(ath0, me:eth)
from_dev0 [0] -> Paint(0) -> grid_demux
from_dev0 [1] -> Paint(0) -> probe_cl :: Classifier(12/7ffe, 12/7ffd);

probe_cl [0] -> ls ->  probe_switch :: Switch(-1) -> dev0;
probe_cl [1] -> ls2 -> probe_switch;

// support for traceroute
dec_ip_ttl :: TTLChecker -> dev0;
dec_ip_ttl [1] -> ip_demux;

grid_demux [0] -> CheckIPHeader( , 82) -> grid_data_demux;
grid_demux [1] -> nb -> dev0;

// not needed in userlevel

arp_demux [1] -> Strip(14)->IPPrint(arp_demux[1]->)->ip_input;
from_dev0 [2] -> ToHost(me);

//ControlSocket(tcp, 7777);

ip_demux [0] -> to_host_encap;  // loopback packet sent by us, required on BSD userlevel
// forward packet sent by us
ip_demux [1] ->ip_to_grid :: GetIPAddress(16) ->
    CheckIPHeader ->
    GridEncap(me:eth, me:ip) ->
    dec_ip_ttl;
```

```
ip_demux [2] -> to_host_encap;  //packets destined for the internal address
ip_demux [3] -> IPPrint(ip_demux[3]->)->ip_to_intern;    //packets destined for the
internal net
// forward packet for the gateway
ip_demux [4] -> GetIPAddress(16) ->
   CheckIPHeader ->
   GridEncap(me:eth, me:ip) ->
   to_gateway ->
   dec_ip_ttl;

grid_data_demux [0] -> Strip(82) -> icmp_packet_classifier :: IPClassifier (icmp type
echo-reply,icmp type echo, proto icmp, -);  //strip the grid header. Determine if this is
an ICMP packet or a TCP/UDP packet
       icmp_packet_classifier[0] -> [1]irw;     // icmp ping reply
       icmp_packet_classifier[1] -> to_host_encap; //icmp ping should go to the kernel
       icmp_packet_classifier[2] -> ierw;     // other icmp
       icmp_packet_classifier[3] -> [3]rewriter;  // receive packet from net for us or
internal network.

// forward packet from grid mesh net for someone else
grid_data_demux [1] -> dec_ip_ttl;
```


from_dev0
FromGridDev
12/7fff
Mesh Data
12/7ffe
12/7ffd
linkstat1&2
other
grid_demux
Classifier
19/03
Data
19/02
Route
Adv.
nb
DSDVRouteTable
CheckIPHeader(, 82)
grid_data_demux
IPClassifier
dst host me
dst net me/24
Strip(82)
dec_ip_ttl
TTLChecker
icmp_packet
_classifier
IPClassifier
icmp type echo-reply
icmp type echo
proto icmp
other
irw
ICMPPingRewriter
to_host_encap
EtherEncap
ierw
ICMPRewriter
rewriter
IPRewriter
ToHost(me)
probe_cl
Classifier
12/7ffe
linkstat1
12/7ffd
linkstat2
ls
LinkStat
ls2
Idle
probe_switch
Switch(-1)
dev0
ToGridDev
to_host_encap
EtherEncap
to_host
ToHost(grid0)
nb
DSDVRouteTable
dev0
ToGridDev
dec_ip_ttl
TTLChecker
no error
error packet
ip_demux
IPClassifier
irw
ICMPPingRewriter
request
reply
ip_demux
IPClassifier
ip_to_intern
GetIPAddress
to_world_pat
to_server_pat
rewriter
IPRewriter
pass
ip_to_exern_class
IPClassifier
dst host intern
dst host me
other
to_host_encap
EtherEncap
rewriter
IPRewriter
pass
ip_to_client_class
IPClassifier
dst host intern
dst host me
other
to_host_encap
EtherEncap
rewriter
IPRewriter
ierw
ICMPRewriter
previous
mapping
no mapping
ip_to_intern
GetIPAddress
to_host_encap
EtherEncap
ip_to_intern
GetIPAddress
CheckIPHeader
intern_arpq
ARPQuerier(intern)
intern_arpq
ARPQuerier(intern)
intern_dev
GatewayDevice(eth0)
ip_demux
IPClassifier
dst host me
dst net me/24
dst host intern
dst net intern/24
other
to_host_encap
EtherEncap
ip_to_grid
GetIPAddress(16)
CheckIPHeader
GridEncap(me:
eth, me:ip)
dec_ip_ttl
TTLChecker
to_host_encap
EtherEncap
ip_to_intern
GetIPAddress
ip_to_grid
GetIPAddress(16)
CheckIPHeader
GridEncap(me:
eth, me:ip)
to_gateway
GridGatewayInfo
dec_ip_ttl
TTLChecker
intern_dev
GatewayDevice
intern_arp_class
Classifier
12/0806 20/0001
arp request
12/0806 20/0002
arp reply
12/0800
data
other
ARPRespon
der(intern)
intern_dev
GatewayDevice
intern_arpq
ARPQuerier(intern)
Strip(14)
intern_data_chk
CheckIPHeader
Discard
icmp_i_packet_
classifier
IPClassifier
icmp type echo
proto icmp
other
irw
ICMPPingRewriter
to_host_encap
EtherEncap
rewriter
IPRewriter
from_host
FromHost
from_host_noGrid
FromHost
arp_demux
Classifier
12/0806 20/0001
arp request
12/0800
data
ARPResponder
(0.0.0/0 1:1:1:1:1:1)
to_host
ToHost(grid0)
Strip(14)
ip_input
CheckIPHeader
GetIPAddress(16)
ip_demux
IPClassifier

dsdv_nat_gw_kernel.click

```
// This file is based on the automatically generated file at Thu Dec 29 18:19:01 CST 2005
with the following command:
// make-dsdv-config.pl -i ath0 -a 192.168.0.3 -u

// the DSDV portion of this configuration performs routing lookup *after* the interface
// queue

//The NAT is based off of the mazu-nat and thomer-nat examples available in the
click/conf/ directory

//One device (ath0) will communicate with the grid mesh network using the DSDV protocol,
while the other
//device (eth0) will communicate with an  external network.
//This router does not implement any packet forwarding or firewall functions


AddressInfo(
//me is the mesh network address of this router
me 192.168.0.3 00:11:95:FF:FD:DE,

//extern is the address of the non-grid device
extern 130.179.130.175 eth0:eth,

//extern_next_hop is the address of the gateway in the external network
extern_next_hop       00:0D:BD:5B:E9:49
);

//external network (non-mesh) gateway device
elementclass GatewayDevice {
  $device |
  from :: FromDevice($device)
        -> output;
  input -> q :: Queue(1024)
        -> to :: ToDevice($device);
  ScheduleInfo(from .1, to 1);
}



elementclass TTLChecker {
  // expects grid packets with MAC headers --- place on output path to
  // decrement the IP TTL for next hop and provide traceroute support.
  //
  // push -> push
```

```
  //
  // output [0] passes through the Grid MAC packets
  //
  // output [1] produces ICMP error packets to be passed back to IP
  // routing layer

  input -> cl :: Classifier(19/03, -);
  cl [1] -> output; // don't try to dec ttl for non-IP packets...

  cl [0]
    -> MarkIPHeader(82)
    -> cl2 :: IPClassifier(src host != me, -);

  cl2 [0]-> dec :: DecIPTTL; // only decrement ttl for packets we don't originate
  cl2 [1] -> output;

  dec [0] -> output;
  dec [1] -> ICMPError(me, 11, 0) -> [1] output;
};

li :: GridLocationInfo2(0, 0, LOC_GOOD false);

elementclass FixupGridHeaders {
  $li | // LocationInfo element
  input
    -> FixSrcLoc($li)
    -> SetGridChecksum
    -> output;
};

elementclass ToGridDev {
  // push, no output
  $dev |
  input -> cl :: Classifier(12/7ffe, // LinkStat 1
                            12/7ffd, // LinkStat 2
                           19/02,
                           19/03);
  prio :: PrioSched;
  cl [0] -> probe_counter :: Counter -> probe_q :: Queue(5) -> [0] prio;
  cl [1] -> probe_counter;
  cl [2] -> route_counter :: Counter -> route_q :: Queue(5) -> FixupGridHeaders(li) ->
[1] prio;
  cl [3] ->  data_counter :: Counter -> data_q :: Queue(5)
    -> data_counter_out :: Counter
    -> tr :: TimeRange
    -> lr :: LookupLocalGridRoute2(me:eth, me:ip, nb)
    -> FixupGridHeaders(li)
    -> data_counter_out2 :: Counter
    -> tr2 :: TimeRange
```

```
    -> [2] prio;
  prio
    -> dev_counter :: Counter
    -> t :: PullTee
    -> ToDevice($dev);
  t [1] -> SetTimestamp -> Discard;
};

elementclass FromGridDev {
  // push, no input
  // `Grid' packets on first output
  // `LinkStat' packets on second output
  $dev, $mac |
  FromDevice($dev, PROMISC false)
    -> t :: Tee
    -> HostEtherFilter($mac, DROP_OWN true)
    -> cl :: Classifier(12/7fff, 12/7ffe, 12/7ffd, -);
  cl [0]  // `Grid' packets
    -> ck :: CheckGridHeader
    -> [0] output;
  cl [1]  // `LinkStat 1' packets
    -> [1] output;
  cl [2]  // `LinkStat 2' packets
    -> [1] output;
  cl [3] // everything else
    -> [2] output;
  t [1] -> Discard;
  ck [1] -> Print('Bad Grid header received', TIMESTAMP true, NBYTES 166) -> Discard;
};

// handles IP packets with no extra encapsulation
ip_demux :: IPClassifier(dst host me,    // ip for us
                         dst net me/24); // ip for Grid network

//input demux from router
arp_demux :: Classifier(12/0806 20/0001, // arp queries
                        12/0800);        // IP packets

//packets input from the router (grid dev)
ip_input :: CheckIPHeader -> GetIPAddress(16) -> ip_demux;

// packets to or from the router its self
//the router classifier may not be necessary
to_host :: ToHost(grid0);
//to_host_noGrid :: ToHost(extern0);
to_host_encap :: EtherEncap(0x0800, 1:1:1:1:1:1, 2:2:2:2:2:2)->to_host;
from_host :: FromHost(grid0, me/24)->arp_demux->ARPResponder(0.0.0/0 1:1:1:1:1:1) ->
to_host;
from_host_noGrid :: FromHost(extern0, extern/24)->arp_demux;
```

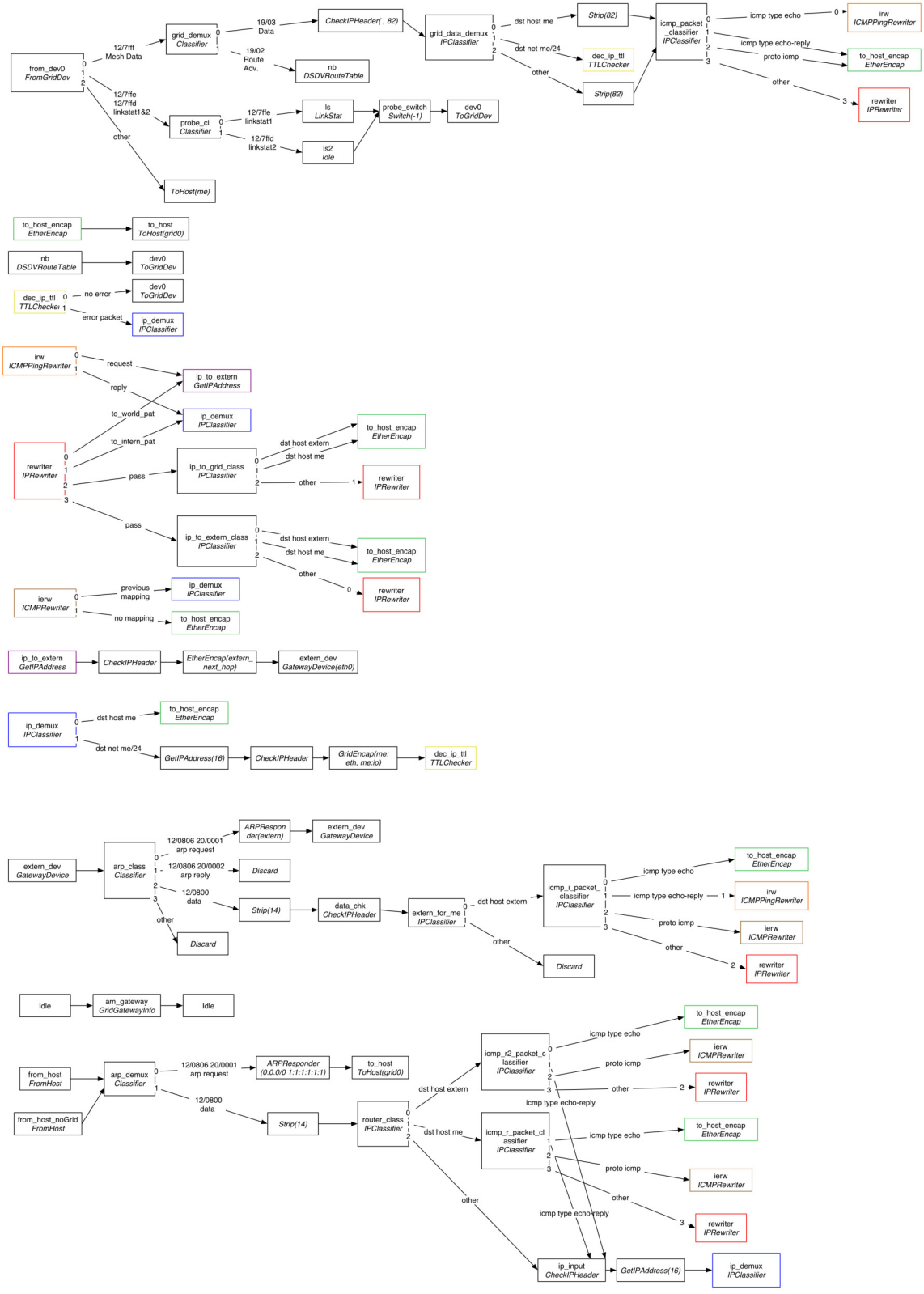
```
router_class :: IPClassifier(dst host extern,dst host me, -);
//classify if the router packets are icmp
icmp_r_packet_classifier :: IPClassifier (icmp type echo,icmp type echo-reply, proto
icmp, -);

icmp_r2_packet_classifier :: IPClassifier (icmp type echo,icmp type echo-reply, proto
icmp, -);

//non-grid device
extern_dev :: GatewayDevice(eth0);


// ARP MACHINERY for non-grid device
arp_class :: Classifier(12/0806 20/0001, 12/0806 20/0002, 12/0800, -);
// ARP Querier
//arpq :: ARPQuerier(extern);

extern_dev -> arp_class;
arp_class[0] -> ARPResponder(extern)   //respond to ARP queries
       -> extern_dev;

//arp_class[1] ->  [1]arpq;
arp_class [1] -> Discard;

//packets from output 2 are data packets (non-ARP). They are dealt with below.


arp_class[3] -> Discard;             //discard unrecognized packets


// REWRITERS for NAT

IPRewriterPatterns(to_world_pat extern 50000-65535 - -,
              to_intern_pat me 50000-65535 - -
              );

rewriter :: IPRewriter(// internal traffic
               pattern to_world_pat 0 1,
               // external traffic
               pattern to_intern_pat 1 0,
               //pass through packets from internal
               //to check for previous routes
               pass 2,
               //pass through packets from external
               //to check for previous routes
               pass 3
);
```

```
// Rewriting rules for ICMP packets
irw :: ICMPPingRewriter(extern, -);

// Rewriting rules for ICMP error packets
ierw :: ICMPRewriter(rewriter irw);

//output path for packets destined for the non-grid network
ip_to_extern :: GetIPAddress(16)
      -> CheckIPHeader
      -> EtherEncap(0x0800, extern:eth, extern_next_hop:eth)
      -> extern_dev;
//ip_to_extern :: GetIPAddress(16)
//      -> CheckIPHeader
//      -> [0]arpq
//      -> extern_dev;

rewriter[3]->ip_to_extern_class :: IPClassifier(dst host extern,dst host me, -);
   ip_to_extern_class[0]->to_host_encap;    //packets addressed to the router
   ip_to_extern_class[1]->to_host_encap;    //packets addressed to the router
   ip_to_extern_class[2]->[0]rewriter;     //packets for the outside

   rewriter[0] -> ip_to_extern;

rewriter[2] -> ip_to_grid_class :: IPClassifier(dst host extern,dst host me, -);
   ip_to_grid_class[0]->to_host_encap;          //packets addressed to the router
   ip_to_grid_class[1]-> to_host_encap;
   ip_to_grid_class[2]-> [1]rewriter;      //packets addressed to another machine go to
the grid mesh network

   rewriter[1] -> ip_demux;



//ICMP Ping packets:
irw[0] -> ip_to_extern;
irw[1] -> ip_demux;

//OTHER ICMP packets
ierw[0] -> ip_demux;
ierw[1] -> to_host_encap;  //packets that were not re-written

//FILTER PACKETS FROM THIS ROUTER
   router_class [0]->icmp_r2_packet_classifier;
   router_class [1]-> icmp_r_packet_classifier;
   router_class [2]-> ip_input;      //other packets

        icmp_r_packet_classifier[0] -> to_host_encap;     // icmp ping
        icmp_r_packet_classifier[1] -> ip_input;   //icmp ping reply
```

```
        icmp_r_packet_classifier[2] -> ierw;   //other icmp
        icmp_r_packet_classifier[3] -> [3]rewriter; // stuff for the mesh network

        icmp_r2_packet_classifier[0] -> to_host_encap;  // icmp ping
        icmp_r2_packet_classifier[1] -> [1]irw;   //icmp ping reply
        icmp_r2_packet_classifier[2] -> ip_to_extern;   //other icmp
        icmp_r2_packet_classifier[3] -> [2]rewriter; // stuff for the mesh network


// FILTER & REWRITE IP PACKETS FROM NON-GRID dev

arp_class[2] -> Strip(14)    //data packets
        -> data_chk :: CheckIPHeader
        -> extern_for_me :: IPClassifier (dst host extern, -);

// Only forward packets destined for us!
extern_for_me [0] -> icmp_i_packet_classifier :: IPClassifier (icmp type echo-reply,icmp
type echo, proto icmp, -);

        icmp_i_packet_classifier[0] -> [1]irw;   //icmp ping reply is re-written
        icmp_i_packet_classifier[1] -> to_host_encap;     // icmp ping
        icmp_i_packet_classifier[2] -> ierw;   //other icmp
        //send packet through [2]rewriter first, to check for previous mappings
        //then check if the packet is for this router
        //if not, send it to the rewriter
         // stuff for the mesh network or for this router
        icmp_i_packet_classifier[3] ->  [2]rewriter;


//packets for other people don't belong in our network
extern_for_me [1] ->Discard;

// Grid mesh network packets

ls2 :: Idle;
ls :: LinkStat(ETH me:eth, SIZE 148 );
metric :: HopcountMetric();

am_gateway :: GridGatewayInfo(nb, true);
//am_gateway will set the ip address of packets to the best gateway
//BUT this is a gateway router
//so this functionality is not needed
Idle->am_gateway->Idle;

nb :: DSDVRouteTable(60000, 15000, 7500, 1000,
                     me:eth, me:ip,
                     MAX_HOPS 100,
                      METRIC metric,
                     GW am_gateway,
```

```
                    VERBOSE true

                     );

grid_demux :: Classifier(19/03,    // encapsulated (data) packets
                         19/02);   // route advertisement packets

// handles IP packets with Grid data encapsulation
grid_data_demux :: IPClassifier(dst host me,    // ip for us
                                dst net me/24, // ip for Grid network
                                -);            //other packets

// dev0 = grid device
dev0 :: ToGridDev(ath0);
from_dev0 :: FromGridDev(ath0, me:eth)
from_dev0 [0] -> Paint(0) -> grid_demux

//linkstat stuff
from_dev0 [1] -> Paint(0) -> probe_cl :: Classifier(12/7ffe, 12/7ffd);

probe_cl [0] -> ls ->  probe_switch :: Switch(-1) -> dev0;
probe_cl [1] -> ls2 -> probe_switch;

// support for traceroute
dec_ip_ttl :: TTLChecker -> dev0;
dec_ip_ttl [1] -> ip_demux;

grid_demux [0] -> CheckIPHeader( , 82) -> grid_data_demux;
grid_demux [1] -> nb -> dev0;

// arp stuff for grid device
// not needed in userlevel
arp_demux [1] -> Strip(14) -> router_class;

from_dev0 [2] -> ToHost(grid0);

// loopback packet sent by us, required on BSD userlevel
ip_demux [0] -> to_host_encap;
// forward packet sent by us
ip_demux [1] ->GetIPAddress(16)->CheckIPHeader-> GridEncap(me:eth, me:ip) -> dec_ip_ttl;


//Determine if this is an ICMP packet or a TCP/UDP packet
grid_data_demux [0] -> Strip(82) -> icmp_packet_classifier :: IPClassifier (icmp type
echo,icmp type echo-reply, proto icmp, -);
       icmp_packet_classifier[0] -> irw;     // icmp ping
       icmp_packet_classifier[1] -> to_host_encap; //icmp ping reply to kernel
       icmp_packet_classifier[2] -> to_host_encap; // other icmp
```

```
        icmp_packet_classifier[3] -> [3]rewriter;  // receive packet from net for us or
internal network.

// forward packet from grid mesh net for someone else
grid_data_demux [1] -> dec_ip_ttl;
//packets destined for the gateway
grid_data_demux [2] -> Strip(82) -> icmp_packet_classifier;
```


from_dev0
FromGridDev
12/7fff
Mesh Data
12/7ffe
12/7ffd
linkstat1&2
other
grid_demux
Classifier
19/03
Data
19/02
Route
Adv.
probe_cl
Classifier
12/7ffe
linkstat1
12/7ffd
linkstat2
ToHost(me)
CheckIPHeader(, 82)
nb
DSDVRouteTable
ls
LinkStat
ls2
Idle
probe_switch
Switch(-1)
dev0
ToGridDev
grid_data_demux
IPClassifier
dst host me
dst net me/24
other
Strip(82)
dec_ip_ttl
TTLChecker
Strip(82)
icmp_packet
_classifier
IPClassifier
icmp type echo
icmp type echo-reply
proto icmp
other
irw
ICMPPingRewriter
to_host_encap
EtherEncap
rewriter
IPRewriter
to_host_encap
EtherEncap
to_host
ToHost(grid0)
nb
DSDVRouteTable
dev0
ToGridDev
dec_ip_ttl
TTLChecker
no error
dev0
ToGridDev
error packet
ip_demux
IPClassifier
irw
ICMPPingRewriter
request
reply
ip_to_extern
GetIPAddress
ip_demux
IPClassifier
rewriter
IPRewriter
to_world_pat
to_intern_pat
pass
ip_to_grid_class
IPClassifier
dst host extern
dst host me
other
to_host_encap
EtherEncap
rewriter
IPRewriter
pass
ip_to_extern_class
IPClassifier
dst host extern
dst host me
other
to_host_encap
EtherEncap
rewriter
IPRewriter
ierw
ICMPRewriter
previous
mapping
no mapping
ip_demux
IPClassifier
to_host_encap
EtherEncap
ip_to_extern
GetIPAddress
CheckIPHeader
EtherEncap(extern_
next_hop)
extern_dev
GatewayDevice(eth0)
ip_demux
IPClassifier
dst host me
dst net me/24
to_host_encap
EtherEncap
GetIPAddress(16)
CheckIPHeader
GridEncap(me:
eth, me:ip)
dec_ip_ttl
TTLChecker
extern_dev
GatewayDevice
arp_class
Classifier
12/0806 20/0001
arp request
12/0806 20/0002
arp reply
12/0800
data
other
ARPRespon
der(extern)
extern_dev
GatewayDevice
Discard
Strip(14)
data_chk
CheckIPHeader
extern_for_me
IPClassifier
dst host extern
other
Discard
icmp_i_packet_
classifier
IPClassifier
icmp type echo
icmp type echo-reply
proto icmp
other
to_host_encap
EtherEncap
irw
ICMPPingRewriter
ierw
ICMPRewriter
rewriter
IPRewriter
Idle
am_gateway
GridGatewayInfo
Idle
from_host
FromHost
from_host_noGrid
FromHost
arp_demux
Classifier
12/0806 20/0001
arp request
12/0800
data
ARPResponder
(0.0.0.0/0 1:1:1:1:1:1)
to_host
ToHost(grid0)
Strip(14)
router_class
IPClassifier
dst host extern
dst host me
other
icmp_r2_packet_c
lassifier
IPClassifier
icmp_r_packet_cl
assifier
IPClassifier
icmp type echo
proto icmp
other
icmp type echo-reply
to_host_encap
EtherEncap
ierw
ICMPRewriter
rewriter
IPRewriter
ip_input
CheckIPHeader
GetIPAddress(16)
ip_demux
IPClassifier

Click License Agreement

(c) 1999-2004 Massachusetts Institute of Technology
(c) 2000-2004 Mazu Networks, Inc.
(c) 2001-2004 International Computer Science Institute
(c) 2004 Regents of the University of California
Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:
The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.
The names and trademarks of copyright holders may not be used in advertising or publicity pertaining to the software without specific prior permission. Title to copyright in this software and any associated documentation will at all times remain with the copyright holders.
THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Appendix C - Automated Startup Code

Meshstart

```
#!/usr/bin/perl

# Check to see if there is a cd-rom drive is mountable and see if there is
# a meshstart.conf file on it.
$cdrom = `cat /proc/sys/dev/cdrom/info|grep \"drive name\"|sed \"s/.*:\\s*//\"`;
chomp($cdrom);
$path_to_config = "/home/mesh/meshstart";

if ($cdrom ne "")
{
   print ("trying to mount cdrom on dev $cdrom");
   $mounted = `mount -t iso9660 \/dev\/$cdrom \/mnt\/cdrom`;
   $rc = $?;
   print ("rc: $rc \n");
   sleep(5);
   if ($rc == 0)
   {
      if (-e "/mnt/cdrom/meshstart.conf")
      {
         $path_to_config = "/mnt/cdrom";
        print ("Using configuration file on cdrom\n");
      }
  }
}

print ("Using $path_to_config/meshstart.conf as config file\n");
require "$path_to_config/meshstart.conf";

if ($cdrom ne "")
{
   $mounted = `umount /mnt/cdrom`;
}

#-----------------------------------------------------------------------#
# Start up script for our thesis project.
#
# Uses meshstart.conf which it assumes is in the same directory
#
# Sets up the following, in order:
#
# 1. configure the wireless cards:
#       - configures the mesh device
#       - if the CLIENT_DEV value is set in the config file, this
```

```
#        card is set to connect clients
# 2. configure click
#      - runs click based on the paramaters in the config file
#      - if CLICK_KERNEL = true, runs click-install, otherwise
#        click (user space) is ran
#      - check to see if the click config file is present or not
#-----------------------------------------------------------------------
#
if ( $MESH_DEV =~ /^ath/ )
{
   `/usr/local/bin/wlanconfig $MESH_DEV destroy`;
}

# Create the commands for setting up the mesh device
@MESH_SETUP = (
   "iwconfig $MESH_DEV mode ad-hoc channel $MESH_CHANNEL essid $MESH_ESSID",
   "ifconfig $MESH_DEV inet $MESH_DEVICE_ADDR netmask $MESH_NETMASK",
   "ifconfig $MESH_DEV up");

# Add the extra line for creating the atheros device
if ( $MESH_DEV =~ /^ath/ )
{
   @MESH_SETUP = ( "/usr/local/bin/wlanconfig $MESH_DEV create wlandev $ATH_WIFI_DEV
wlanmode ibss",
      @MESH_SETUP);
}

# If this node holds a wireless card to connect clients, create the setup commands.
if ($CLIENT_DEV ne "")
{
   @CLIENT_SETUP = (
      "ifconfig $CLIENT_DEV inet $CLIENT_DEVICE_ADDR netmask $CLIENT_NETMASK",
      "ifconfig $CLIENT_DEV up");

   if ($CLIENT_CHANNEL ne "")
   {
     @CLIENT_SETUP = ("iwconfig $CLIENT_DEV mode ad-hoc channel $CLIENT_CHANNEL essid
$CLIENT_ESSID",
     @CLIENT_SETUP);
   }
}

# to start with, lets let 'em know we're running, beep twice:
$date = `date`;
print ("Starting meshstart:  $date\n") ;
system ($BEEP_STARTUP);

#-----------------------------------------------------------------------#
# Configure mesh device                                                 #
```

```
#-------------------------------------------------------------------------#
if ( $MESH_DEV ne "" )
{
   print ("Setting up $MESH_DEV\n");
   foreach $cmd (@MESH_SETUP)
   {
      print "Running: $cmd\n";
      `$cmd`;
      if ($? != 0)
      {
         print ("Couldn't execute: $cmd\n $1");
         sleep(2);
         system($BEEP_FAILURE);
         die;
      }
   }
   sleep(2);
   system($BEEP_MESH_SETUP);
}
else
{
   print ("If you're not setting up a mesh device, why run this file?\n");
   system($BEEP_FAILURE);
   die;
}

#-------------------------------------------------------------------------#
# Configure client device if needed                                       #
#-------------------------------------------------------------------------
#
if ( $CLIENT_DEV ne "" )
{
   print ("Setting up $CLIENT_DEV as a client interface\n");
   foreach $cmd (@CLIENT_SETUP)
   {
      print "Running: $cmd\n";
      `$cmd`;
      if ( $? != 0 )
      {
         print ("Couldn't execute: $cmd\n $1");
         sleep(2);
         system($BEEP_FAILURE);
         die;
      }
   }
   sleep(2);
   system($BEEP_CLIENT_SETUP);
}
else
```

```
{
   print ("\nNo CLIENT_DEV info found, not setting up client network\n\n");
}

#--------------------------------------------------------------------------#
# Start Click                                                              #
#--------------------------------------------------------------------------#

if ( $CLICK_KERNEL ne "" && -e $CLICK_CONFIG )
{
   if ($CLICK_KERNEL eq "true" )
   {
      $click_prog = "click-install ";
   }
   else
   {
      $click_prog = "click ";
   }
   $cmd = $click_prog . $CLICK_CONFIG;
   if ($CLICK_KERNEL eq "true")
   {
       print "Running: $cmd \n";
      `$cmd`;
   }
   else
   {
      $rc = `which $click_prog`;
      if ($? == 0)
      {
          print "Starting: $cmd in backgroud\n";
         system("$cmd &");
      }
      else
      {
          print ("Couldn't execute: $cmd\n $1");
         sleep(2);
         system($BEEP_FAILURE);
         die;
      }
   }
   if ( $? != 0 )
   {
      print ("Couldn't execute: $cmd\n $1");
      sleep(2);
      system($BEEP_FAILURE);
      die;
    }
    sleep(2);
    system($BEEP_CLICK_START);
```

```
}
else
{
   print("Can't figure out click configuration parameters (didn't try too hard)\n");
   sleep (2);
   system($BEEP_FAILURE);
}

$date = `date`;
print ("Finished meshstart:  $date\n") ;
```

Meshstart.conf

```
#--------------------------------------------------------------------#
# Define beeps                                                       #
# -------------------------------------------------------------------#
$BEEP_STARTUP = "beep -f 1200;beep -f 1300;beep -f 1500;beep -f 1700";
$BEEP_FAILURE = "beep -f 1000 -l 300;beep -f 100 -l 700";

$BEEP_MESH_SETUP = "beep -f 600;";
$BEEP_CLIENT_SETUP = "beep -f 1000;beep -f 1000";
$BEEP_NETWORKCARD_SETUP = "beep -f 1200;beep -f 1200;beep -f 1200";

$BEEP_CLICK_START = "beep -f 400 -l 300;beep -f 700;beep -f 1200 -l 300;beep -f 700; beep
-f 400 -l 300";

#--------------------------------------------------------------------#
# Define Mesh Interface                                              #
# -------------------------------------------------------------------#
$MESH_CHANNEL = "30";
$MESH_ESSID = "meshnet";
$MESH_DEV = "ath0";
$MESH_DEVICE_ADDR = "10.0.0.3";
$MESH_NETMASK = "255.255.255.0";
# Use this value if using an atheros device for the mesh
$ATH_WIFI_DEV = "wifi0";

#--------------------------------------------------------------------#
# Define Client Interface                                            #
# -------------------------------------------------------------------#
#$CLIENT_CHANNEL = "11";
#$CLIENT_ESSID = "clientnet";
$CLIENT_DEV = "eth0";
$CLIENT_DEVICE_ADDR = "11.0.0.3";
```

```
$CLIENT_NETMASK = "255.255.255.0";

#--------------------------------------------------------------------#
# Define Click Parameters                                            #
# -------------------------------------------------------------------#
$CLICK_KERNEL = "true";
$CLICK_CONFIG = "/tmp/dsdv_nat_gw_kernel.click";
```

Memwatch

```
#!/usr/bin/perl

# Watch the memory and restart click/meshnet when low
$mem_cmd = "cat \/proc\/meminfo | grep MemFree | sed \"s\/MemFree:\\s*\/\/\"| sed
\"s\/\\s*kB\/\/\"";
$mem_free = `$mem_cmd`;
#print ("$mem_free\n");
$date = `date`;
chomp ($date);
print ("$date:  memwatch starting\n");
while (1)
{
   if ($mem_free < 5000)
   {
      $date = `date`;
      chomp ($date);
      print ("$date: Restarting mesh network, low mem!!");
      exec ("/etc/init.d/meshstart restart");
   }
   sleep(5);
   $mem_free = `$mem_cmd`;
}
```

Meshstart – init script

```
#!/sbin/runscript

start(){
        ebegin "Starting mesh network"
        /home/mesh/meshstart/meshstart >> /var/log/meshstart
        /home/mesh/meshstart/memwatch >> /var/log/memwatch &
        eend $? "Error starting meshnetwork"
}

stop(){
        ebegin "Stopping mesh network"
        /home/mesh/meshstart/meshstop >> /var/log/meshstart
```

```
        eend $? "Error stopping meshnetwork"
}

restart(){
        ebegin "Restarting mesh network"
        /home/mesh/meshstart/meshstop >> /var/log/meshstart
        /home/mesh/meshstart/meshstart >> /var/log/meshstart
        /home/mesh/meshstart/memwatch >> /var/log/memwatch &
        eend $? "Error restarting meshnetwork"

}
```

Appendix D - Click Modular Router Output

TCP Mappings from client gateway router

```
(192.168.1.13, 44829, 192.168.0.3, 22) => (192.168.0.1, 63989, 192.168.0.3, 22) [0]
(192.168.1.13, 44828, 192.168.0.3, 22) => (192.168.0.1, 56490, 192.168.0.3, 22) [0]
```

ICMP Mapping from client gateway router

```
(192.168.1.13, 192.168.0.4, 22038) => (192.168.0.1, 192.168.0.4, 32781)
```

DSDV Route Table Output from Click Modular Router for a Complete System in a Linear Topology

```
From Mesh 1: 192.168.0.1
192.168.0.3 next=192.168.0.3 hops=1 gw=y
192.168.0.2 next=192.168.0.2 hops=1 gw=n
192.168.0.4 next=192.168.0.2 hops=2 gw=n
192.168.0.5 next=192.168.0.2 hops=3 gw=n

From Mesh 2: 192.168.0.2
192.168.0.3 next=192.168.0.1 hops=2 gw=y
192.168.0.1 next=192.168.0.1 hops=1 gw=n
192.168.0.4 next=192.168.0.4 hops=1 gw=n
192.168.0.5 next=192.168.0.4 hops=2 gw=n

From Mesh 3: 192.168.0.3
192.168.0.1 next=192.168.0.2 hops=1 gw=n
192.168.0.2 next=192.168.0.2 hops=2 gw=n
192.168.0.4 next=192.168.0.2 hops=3 gw=n
192.168.0.5 next=192.168.0.2 hops=4 gw=n

From Mesh 4: 192.168.0.4
192.168.0.3 next=192.168.0.2 hops=3 gw=y
192.168.0.1 next=192.168.0.2 hops=2 gw=n
192.168.0.2 next=192.168.0.2 hops=1 gw=n
192.168.0.5 next=192.168.0.5 hops=1 gw=n

From Mesh 5: 192.168.0.5
192.168.0.3 next=192.168.0.4 hops=4 gw=y
192.168.0.1 next=192.168.0.4 hops=3 gw=n
192.168.0.2 next=192.168.0.4 hops=2 gw=n
192.168.0.4 next=192.168.0.4 hops=1 gw=n
```

Appendix E - Budget

All hardware was acquired except for the two 802.11b/g wireless cards that would connect clients to the mesh network. These cards have a budgeted price of $77.00 and were not purchased as a 802.11b card was available for free. As can be seen in the table below, the total cost of acquired parts is below the total budgeted price, and as such, the project should come under budget as a whole. These prices include all applicable taxes.

Table: Budget Comparison of Purchased Hardware

Part	***Budgeted Cost per Unit***	***Actual Cost per Unit***	***Budgeted Total Cost***	***Total Cost***	***Acquired Through***
DWL-AG30 x 5	$140.00	$92.60	$700.00	$463.00	Department of ECE
Wireless 802.11b/g x 2	$77.00	TBD	154.00	TBD	Department of ECE
Ethernet Card (Loaned From Tech. Shop) x 1	$0.00	$30.00	$0.00	$30.00	Department of ECE
6' Ethernet Cable x 1	$8.00	$8.00	$56.00	$8.00	Department of ECE
Desktop computer x 5	$0.00	$0.00	$0.00	$0.00	Internet Innovation Center
Computer display Screen	$0.00	$0.00	$0.00	$0.00	Department of ECE
Linux Operating System	$0.00	$0.00	$0.00	$0.00	Department of ECE
Keyboard & mouse	$0.00	$0.00	$0.00	$0.00	Internet Innovation Center
Totals			$910.00	$501.00	**Difference: $409.00**

Appendix F - Raw Testing Output

	Test 1	test 2	Test 3	Test 4	Test 5	**Average**
TCP Test						
Throughput (Bps)						
Hop 1	199289	202972	201958	205914	198480	201722.6
Hop 2	157626	253161	224806	209338	224906	213967.4
Hop 3	154531	137546	152179	163190	191229	159735
Avg. Pkt Delay						
Hop 1	0.0005	0.0006	0.0005	0.0005	0.0006	0.000540
Hop 2	0.0014	0.0001	0.0002	0.0003	0.0003	0.000460
Hop 3	0.0002	0.0003	0.0005	0.0004	0.0003	0.000340
UDP Test						
Throughput (Bps)						
Hop 1	102408	102421	102421	102420	101175	102169
Hop 2	40974	28500	32758	39762	67606	41920
Hop 3	69130	97253	98938	97461	99042	92365
Avg. Pkt Delay (s)						
Hop 1	0.01004	0.01015	0.010213	0.01002	0.024279	0.0129404
Hop 2	0.0964	0.119496	0.106213	0.106857	0.069913	0.0997756
Hop 3	0.01335	0.014594	0.014941	0.014395	0.015609	0.0145768

```
============================================================
./1hop/itgsend_1hop_test1_udp.log
============================================================
------------------------------------------------------------
Flow number: 1
From 192.168.1.13:32782
To    130.179.130.2:8999
------------------------------------------------------------
Total time                =     29.997578 s
Total packets             =          6000
Minimum delay             =      0.003947 s
Maximum delay             =      0.135845 s
Average delay             =      0.010045 s
Average jitter            =      0.001959 s
Delay standard deviation  =      0.002883 s
Bytes received            =       3072000
Average bitrate           =    819.266142 Kbit/s
Average packet rate       =    200.016148 pkt/s
Packets dropped           =             0 (0.00 %)
------------------------------------------------------------

************************  TOTAL RESULTS   ******************

Number of flows           =             1
Total time                =     29.997578 s
Total packets             =          6000
Minimum delay             =      0.003947 s
Maximum delay             =      0.135845 s
Average delay             =      0.010045 s
```

```
Average jitter           =      0.001959 s
Delay standard deviation =      0.002883 s
Bytes received           =       3072000
Average bitrate          =    819.266142 Kbit/s
Average packet rate      =    200.016148 pkt/s
Packets dropped          =             0 (0.00 %)
Error lines              =             0
-------------------------------------------------------------
=============================================================================
./1hop/itgsend_1hop_test2_udp.log
=============================================================================
-------------------------------------------------------------
Flow number: 1
From 192.168.1.13:32783
To    130.179.130.2:8999
-------------------------------------------------------------
Total time               =     29.993872 s
Total packets            =          6000
Minimum delay            =      0.003741 s
Maximum delay            =      0.488786 s
Average delay            =      0.010150 s
Average jitter           =      0.002087 s
Delay standard deviation =      0.008933 s
Bytes received           =       3072000
Average bitrate          =    819.367369 Kbit/s
Average packet rate      =    200.040862 pkt/s
Packets dropped          =             0 (0.00 %)
-------------------------------------------------------------

****************  TOTAL RESULTS   ******************

Number of flows          =             1
Total time               =     29.993872 s
Total packets            =          6000
Minimum delay            =      0.003741 s
Maximum delay            =      0.488786 s
Average delay            =      0.010150 s
Average jitter           =      0.002087 s
Delay standard deviation =      0.008933 s
Bytes received           =       3072000
Average bitrate          =    819.367369 Kbit/s
Average packet rate      =    200.040862 pkt/s
Packets dropped          =             0 (0.00 %)
Error lines              =             0
-------------------------------------------------------------
=============================================================================
./1hop/itgsend_1hop_test3_udp.log
=============================================================================
-------------------------------------------------------------
```

```
Flow number: 1
From 192.168.1.13:32783
To    130.179.130.2:8999
------------------------------------------------------------
Total time                =     29.993815 s
Total packets             =          6000
Minimum delay             =      0.007034 s
Maximum delay             =      0.597276 s
Average delay             =      0.010213 s
Average jitter            =      0.002080 s
Delay standard deviation  =      0.010860 s
Bytes received            =       3072000
Average bitrate           =    819.368927 Kbit/s
Average packet rate       =    200.041242 pkt/s
Packets dropped           =             0 (0.00 %)
------------------------------------------------------------

____________________________________________________________
****************  TOTAL RESULTS   ******************

____________________________________________________________
Number of flows           =             1
Total time                =     29.993815 s
Total packets             =          6000
Minimum delay             =      0.007034 s
Maximum delay             =      0.597276 s
Average delay             =      0.010213 s
Average jitter            =      0.002080 s
Delay standard deviation  =      0.010860 s
Bytes received            =       3072000
Average bitrate           =    819.368927 Kbit/s
Average packet rate       =    200.041242 pkt/s
Packets dropped           =             0 (0.00 %)
Error lines               =             0
------------------------------------------------------------

============================================================================
./1hop/itgsend_1hop_test4_udp.log
============================================================================
------------------------------------------------------------
Flow number: 1
From 192.168.1.13:32783
To    130.179.130.2:8999
------------------------------------------------------------
Total time                =     29.994043 s
Total packets             =          6000
Minimum delay             =      0.003682 s
Maximum delay             =      0.032676 s
Average delay             =      0.010020 s
Average jitter            =      0.001989 s
Delay standard deviation  =      0.001827 s
```

```
Bytes received           =       3072000
Average bitrate          =    819.362698 Kbit/s
Average packet rate      =    200.039721 pkt/s
Packets dropped          =             0 (0.00 %)
------------------------------------------------------------

____________________________________________________________
****************  TOTAL RESULTS   ******************

____________________________________________________________
Number of flows          =             1
Total time               =     29.994043 s
Total packets            =          6000
Minimum delay            =      0.003682 s
Maximum delay            =      0.032676 s
Average delay            =      0.010020 s
Average jitter           =      0.001989 s
Delay standard deviation =      0.001827 s
Bytes received           =       3072000
Average bitrate          =    819.362698 Kbit/s
Average packet rate      =    200.039721 pkt/s
Packets dropped          =             0 (0.00 %)
Error lines              =             0
------------------------------------------------------------
============================================================================
./1hop/itgsend_1hop_test5_udp.log
============================================================================
------------------------------------------------------------
Flow number: 1
From 192.168.1.13:32783
To    130.179.130.2:8999
------------------------------------------------------------
Total time               =     29.993901 s
Total packets            =          5927
Minimum delay            =      0.003663 s
Maximum delay            =      0.402134 s
Average delay            =      0.024279 s
Average jitter           =      0.002072 s
Delay standard deviation =      0.015218 s
Bytes received           =       3034624
Average bitrate          =    809.397617 Kbit/s
Average packet rate      =    197.606840 pkt/s
Packets dropped          =            73 (1.22 %)
------------------------------------------------------------


____________________________________________________________
****************  TOTAL RESULTS   ******************

____________________________________________________________
Number of flows          =             1
Total time               =     29.993901 s
Total packets            =          5927
```

```
Minimum delay            =      0.003663 s
Maximum delay            =      0.402134 s
Average delay            =      0.024279 s
Average jitter           =      0.002072 s
Delay standard deviation =      0.015218 s
Bytes received           =       3034624
Average bitrate          =    809.397617 Kbit/s
Average packet rate      =    197.606840 pkt/s
Packets dropped          =            73 (1.22 %)
Error lines              =             0
-------------------------------------------------------------
=============================================================================
./1hop/itgsend_1hop_test6_udp.log
=============================================================================
-------------------------------------------------------------
Flow number: 1
From 192.168.1.13:32783
To    130.179.130.2:8999
-------------------------------------------------------------
Total time               =     29.993858 s
Total packets            =          6000
Minimum delay            =      0.003903 s
Maximum delay            =      0.639861 s
Average delay            =      0.015030 s
Average jitter           =      0.001813 s
Delay standard deviation =      0.014105 s
Bytes received           =       3072000
Average bitrate          =    819.367752 Kbit/s
Average packet rate      =    200.040955 pkt/s
Packets dropped          =             0 (0.00 %)
-------------------------------------------------------------

****************  TOTAL RESULTS   ******************

Number of flows          =             1
Total time               =     29.993858 s
Total packets            =          6000
Minimum delay            =      0.003903 s
Maximum delay            =      0.639861 s
Average delay            =      0.015030 s
Average jitter           =      0.001813 s
Delay standard deviation =      0.014105 s
Bytes received           =       3072000
Average bitrate          =    819.367752 Kbit/s
Average packet rate      =    200.040955 pkt/s
Packets dropped          =             0 (0.00 %)
Error lines              =             0
-------------------------------------------------------------
=============================================================================
```

```
./2hop/itgsend_2hop_test1_udp.log
============================================================================
------------------------------------------------------------
Flow number: 1
From 192.168.1.13:32796
To    130.179.130.2:8999
------------------------------------------------------------
Total time               =     30.077295 s
Total packets            =          2407
Minimum delay            =      0.007322 s
Maximum delay            =      0.275417 s
Average delay            =      0.096399 s
Average jitter           =      0.009193 s
Delay standard deviation =      0.047553 s
Bytes received           =       1232384
Average bitrate          =    327.791179 Kbit/s
Average packet rate      =     80.027143 pkt/s
Packets dropped          =          3592 (59.88 %)
------------------------------------------------------------


____________________________________________________________
****************  TOTAL RESULTS   ******************
____________________________________________________________
Number of flows          =             1
Total time               =     30.077295 s
Total packets            =          2407
Minimum delay            =      0.007322 s
Maximum delay            =      0.275417 s
Average delay            =      0.096399 s
Average jitter           =      0.009193 s
Delay standard deviation =      0.047553 s
Bytes received           =       1232384
Average bitrate          =    327.791179 Kbit/s
Average packet rate      =     80.027143 pkt/s
Packets dropped          =          3592 (59.88 %)
Error lines              =             0
------------------------------------------------------------
============================================================================
./2hop/itgsend_2hop_test2_udp.log
============================================================================
------------------------------------------------------------
Flow number: 1
From 192.168.1.13:32796
To    130.179.130.2:8999
------------------------------------------------------------
Total time               =     30.008821 s
Total packets            =          1920
Minimum delay            =      0.032241 s
Maximum delay            =      0.634187 s
```

```
Average delay            =      0.119496 s
Average jitter           =      0.014578 s
Delay standard deviation =      0.055969 s
Bytes received           =        983040
Average bitrate          =    262.066944 Kbit/s
Average packet rate      =     63.981187 pkt/s
Packets dropped          =          4073 (67.96 %)
-------------------------------------------------------------

_____________________________________________________________

****************  TOTAL RESULTS   ******************

_____________________________________________________________

Number of flows          =             1
Total time               =     30.008821 s
Total packets            =          1920
Minimum delay            =      0.032241 s
Maximum delay            =      0.634187 s
Average delay            =      0.119496 s
Average jitter           =      0.014578 s
Delay standard deviation =      0.055969 s
Bytes received           =        983040
Average bitrate          =    262.066944 Kbit/s
Average packet rate      =     63.981187 pkt/s
Packets dropped          =          4073 (67.96 %)
Error lines              =             0
-------------------------------------------------------------
=============================================================================
./2hop/itgsend_2hop_test3_udp.log
=============================================================================
-------------------------------------------------------------
Flow number: 1
From 192.168.1.13:32796
To    130.179.130.2:8999
-------------------------------------------------------------
Total time               =     30.000888 s
Total packets            =          1670
Minimum delay            =      0.023159 s
Maximum delay            =      0.867384 s
Average delay            =      0.106213 s
Average jitter           =      0.014401 s
Delay standard deviation =      0.049626 s
Bytes received           =        855040
Average bitrate          =    228.003918 Kbit/s
Average packet rate      =     55.665019 pkt/s
Packets dropped          =          4325 (72.14 %)
-------------------------------------------------------------

_____________________________________________________________

****************  TOTAL RESULTS   ******************

_____________________________________________________________

Number of flows          =             1
```

```
Total time               =     30.000888 s
Total packets            =          1670
Minimum delay            =      0.023159 s
Maximum delay            =      0.867384 s
Average delay            =      0.106213 s
Average jitter           =      0.014401 s
Delay standard deviation =      0.049626 s
Bytes received           =        855040
Average bitrate          =    228.003918 Kbit/s
Average packet rate      =     55.665019 pkt/s
Packets dropped          =          4325 (72.14 %)
Error lines              =             0
-------------------------------------------------------------
=============================================================================
./2hop/itgsend_2hop_test4_udp.log
=============================================================================
-------------------------------------------------------------
Flow number: 1
From 192.168.1.13:33269
To    130.179.130.2:8999
-------------------------------------------------------------
Total time               =     30.002699 s
Total packets            =          2330
Minimum delay            =      0.023985 s
Maximum delay            =      0.353815 s
Average delay            =      0.106857 s
Average jitter           =      0.012471 s
Delay standard deviation =      0.037821 s
Bytes received           =       1192960
Average bitrate          =    318.094049 Kbit/s
Average packet rate      =     77.659680 pkt/s
Packets dropped          =          3669 (61.16 %)
-------------------------------------------------------------

****************  TOTAL RESULTS   ******************

Number of flows          =             1
Total time               =     30.002699 s
Total packets            =          2330
Minimum delay            =      0.023985 s
Maximum delay            =      0.353815 s
Average delay            =      0.106857 s
Average jitter           =      0.012471 s
Delay standard deviation =      0.037821 s
Bytes received           =       1192960
Average bitrate          =    318.094049 Kbit/s
Average packet rate      =     77.659680 pkt/s
Packets dropped          =          3669 (61.16 %)
Error lines              =             0
```

```
------------------------------------------------------------
============================================================================
./2hop/itgsend_2hop_test5_udp.log
============================================================================
------------------------------------------------------------
Flow number: 1
From 192.168.1.13:33269
To    130.179.130.2:8999
------------------------------------------------------------
Total time                =     29.990074 s
Total packets             =          3960
Minimum delay             =      0.007983 s
Maximum delay             =      1.012403 s
Average delay             =      0.069913 s
Average jitter            =      0.006527 s
Delay standard deviation =      0.051335 s
Bytes received            =       2027520
Average bitrate           =    540.850950 Kbit/s
Average packet rate       =    132.043689 pkt/s
Packets dropped           =          2040 (34.00 %)
------------------------------------------------------------

____________________________________________________________
****************  TOTAL RESULTS   ******************

____________________________________________________________
Number of flows           =             1
Total time                =     29.990074 s
Total packets             =          3960
Minimum delay             =      0.007983 s
Maximum delay             =      1.012403 s
Average delay             =      0.069913 s
Average jitter            =      0.006527 s
Delay standard deviation =      0.051335 s
Bytes received            =       2027520
Average bitrate           =    540.850950 Kbit/s
Average packet rate       =    132.043689 pkt/s
Packets dropped           =          2040 (34.00 %)
Error lines               =             0
------------------------------------------------------------
============================================================================
./3hop/itgsend_3hop_test1_udp.log
============================================================================
------------------------------------------------------------
Flow number: 1
From 192.168.1.13:33272
To    130.179.130.2:8999
------------------------------------------------------------
Total time                =     26.045652 s
Total packets             =          5033
Minimum delay             =      0.006836 s
```

```
Maximum delay            =      0.024891 s
Average delay            =      0.013345 s
Average jitter           =      0.001787 s
Delay standard deviation =      0.003078 s
Bytes received           =       2576896
Average bitrate          =    791.501322 Kbit/s
Average packet rate      =    193.237628 pkt/s
Packets dropped          =           176 (3.38 %)
-------------------------------------------------------------

_____________________________________________________________
****************  TOTAL RESULTS   ******************
_____________________________________________________________
Number of flows          =             1
Total time               =     26.045652 s
Total packets            =          5033
Minimum delay            =      0.006836 s
Maximum delay            =      0.024891 s
Average delay            =      0.013345 s
Average jitter           =      0.001787 s
Delay standard deviation =      0.003078 s
Bytes received           =       2576896
Average bitrate          =    791.501322 Kbit/s
Average packet rate      =    193.237628 pkt/s
Packets dropped          =           176 (3.38 %)
Error lines              =             0
-------------------------------------------------------------
=====================================================================
./3hop/itgsend_3hop_test2_udp.log
=====================================================================
-------------------------------------------------------------
Flow number: 1
From 192.168.1.13:33274
To    130.179.130.2:8999
-------------------------------------------------------------
Total time               =     29.995683 s
Total packets            =          4050
Minimum delay            =      0.002227 s
Maximum delay            =      1.000425 s
Average delay            =      0.014395 s
Average jitter           =      0.002281 s
Delay standard deviation =      0.022118 s
Bytes received           =       2073600
Average bitrate          =    553.039582 Kbit/s
Average packet rate      =    135.019429 pkt/s
Packets dropped          =          1950 (32.50 %)
-------------------------------------------------------------

_____________________________________________________________
****************  TOTAL RESULTS   ******************
_____________________________________________________________
```

```
Number of flows          =             1
Total time               =     29.995683 s
Total packets            =          4050
Minimum delay            =      0.002227 s
Maximum delay            =      1.000425 s
Average delay            =      0.014395 s
Average jitter           =      0.002281 s
Delay standard deviation =      0.022118 s
Bytes received           =       2073600
Average bitrate          =    553.039582 Kbit/s
Average packet rate      =    135.019429 pkt/s
Packets dropped          =          1950 (32.50 %)
Error lines              =             0
---------------------------------------------------------
=========================================================================
./3hop/itgsend_3hop_test3_udp.log
=========================================================================
---------------------------------------------------------
Flow number: 1
From 192.168.1.13:33274
To    130.179.130.2:8999
---------------------------------------------------------
Total time               =     18.977654 s
Total packets            =          2404
Minimum delay            =      0.007079 s
Maximum delay            =      0.032014 s
Average delay            =      0.013977 s
Average jitter           =      0.001863 s
Delay standard deviation =      0.003080 s
Bytes received           =       1230848
Average bitrate          =    518.862026 Kbit/s
Average packet rate      =    126.675299 pkt/s
Packets dropped          =          1391 (36.65 %)
---------------------------------------------------------
_________________________________________________________
****************  TOTAL RESULTS   ******************

_________________________________________________________
Number of flows          =             1
Total time               =     18.977654 s
Total packets            =          2404
Minimum delay            =      0.007079 s
Maximum delay            =      0.032014 s
Average delay            =      0.013977 s
Average jitter           =      0.001863 s
Delay standard deviation =      0.003080 s
Bytes received           =       1230848
Average bitrate          =    518.862026 Kbit/s
Average packet rate      =    126.675299 pkt/s
Packets dropped          =          1391 (36.65 %)
```

```
Error lines              =             0
-------------------------------------------------------------
=============================================================================
./3hop/itgsend_3hop_test4_udp.log
=============================================================================
-------------------------------------------------------------
Flow number: 1
From 192.168.1.13:33275
To    130.179.130.2:8999
-------------------------------------------------------------
Total time               =     21.720440 s
Total packets            =          4141
Minimum delay            =      0.006405 s
Maximum delay            =      0.077029 s
Average delay            =      0.013551 s
Average jitter           =      0.001933 s
Delay standard deviation =      0.003689 s
Bytes received           =       2120192
Average bitrate          =    780.902044 Kbit/s
Average packet rate      =    190.649913 pkt/s
Packets dropped          =           203 (4.67 %)
-------------------------------------------------------------
_____________________________________________________________
****************  TOTAL RESULTS   ******************

_____________________________________________________________
Number of flows          =             1
Total time               =     21.720440 s
Total packets            =          4141
Minimum delay            =      0.006405 s
Maximum delay            =      0.077029 s
Average delay            =      0.013551 s
Average jitter           =      0.001933 s
Delay standard deviation =      0.003689 s
Bytes received           =       2120192
Average bitrate          =    780.902044 Kbit/s
Average packet rate      =    190.649913 pkt/s
Packets dropped          =           203 (4.67 %)
Error lines              =             0
-------------------------------------------------------------
=============================================================================
./3hop/itgsend_3hop_test5_udp.log
=============================================================================
-------------------------------------------------------------
Flow number: 1
From 192.168.1.13:32769
To    130.179.130.2:8999
-------------------------------------------------------------
Total time               =     29.998751 s
Total packets            =          5803
```

```
Minimum delay            =      0.007100 s
Maximum delay            =      1.004630 s
Average delay            =      0.014941 s
Average jitter           =      0.002154 s
Delay standard deviation =      0.018574 s
Bytes received           =       2971136
Average bitrate          =    792.335921 Kbit/s
Average packet rate      =    193.441387 pkt/s
Packets dropped          =           197 (3.28 %)
---------------------------------------------------------

_________________________________________________________

****************  TOTAL RESULTS   ******************

_________________________________________________________

Number of flows          =             1
Total time               =     29.998751 s
Total packets            =          5803
Minimum delay            =      0.007100 s
Maximum delay            =      1.004630 s
Average delay            =      0.014941 s
Average jitter           =      0.002154 s
Delay standard deviation =      0.018574 s
Bytes received           =       2971136
Average bitrate          =    792.335921 Kbit/s
Average packet rate      =    193.441387 pkt/s
Packets dropped          =           197 (3.28 %)
Error lines              =             0
---------------------------------------------------------
=========================================================================
./3hop/itgsend_3hop_test6_udp.log
=========================================================================
---------------------------------------------------------
Flow number: 1
From 192.168.1.13:32769
To    130.179.130.2:8999
---------------------------------------------------------
Total time               =     20.229251 s
Total packets            =          3975
Minimum delay            =      0.006441 s
Maximum delay            =      0.040052 s
Average delay            =      0.014776 s
Average jitter           =      0.001771 s
Delay standard deviation =      0.002618 s
Bytes received           =       2035200
Average bitrate          =    804.854317 Kbit/s
Average packet rate      =    196.497636 pkt/s
Packets dropped          =            70 (1.73 %)
---------------------------------------------------------

_________________________________________________________

****************  TOTAL RESULTS   ******************
```

```
------------------------------------------------------------
Number of flows          =             1
Total time               =     20.229251 s
Total packets            =          3975
Minimum delay            =      0.006441 s
Maximum delay            =      0.040052 s
Average delay            =      0.014776 s
Average jitter           =      0.001771 s
Delay standard deviation =      0.002618 s
Bytes received           =       2035200
Average bitrate          =    804.854317 Kbit/s
Average packet rate      =    196.497636 pkt/s
Packets dropped          =            70 (1.73 %)
Error lines              =             0
------------------------------------------------------------
============================================================================
./3hop/itgsend_3hop_test7_udp.log
============================================================================
------------------------------------------------------------
Flow number: 1
From 192.168.1.13:32769
To    130.179.130.2:8999
------------------------------------------------------------
Total time               =     26.172406 s
Total packets            =          4982
Minimum delay            =      0.005815 s
Maximum delay            =      0.100058 s
Average delay            =      0.014594 s
Average jitter           =      0.001896 s
Delay standard deviation =      0.004645 s
Bytes received           =       2550784
Average bitrate          =    779.686514 Kbit/s
Average packet rate      =    190.353153 pkt/s
Packets dropped          =           249 (4.76 %)
------------------------------------------------------------

************************************************************
****************  TOTAL RESULTS   ******************
************************************************************
Number of flows          =             1
Total time               =     26.172406 s
Total packets            =          4982
Minimum delay            =      0.005815 s
Maximum delay            =      0.100058 s
Average delay            =      0.014594 s
Average jitter           =      0.001896 s
Delay standard deviation =      0.004645 s
Bytes received           =       2550784
Average bitrate          =    779.686514 Kbit/s
Average packet rate      =    190.353153 pkt/s
```

```
Packets dropped          =           249 (4.76 %)
Error lines              =             0
-------------------------------------------------------------
=============================================================================
./3hop/itgsend_3hop_test8_udp.log
=============================================================================
-------------------------------------------------------------
Flow number: 1
From 192.168.1.13:32795
To    130.179.130.2:8999
-------------------------------------------------------------
Total time               =     23.485464 s
Total packets            =          4461
Minimum delay            =      0.005605 s
Maximum delay            =      0.092024 s
Average delay            =      0.015609 s
Average jitter           =      0.001879 s
Delay standard deviation =      0.007593 s
Bytes received           =       2284032
Average bitrate          =    778.024058 Kbit/s
Average packet rate      =    189.947280 pkt/s
Packets dropped          =           236 (5.02 %)
-------------------------------------------------------------
_____________________________________________________________
****************  TOTAL RESULTS   ******************

_____________________________________________________________
Number of flows          =             1
Total time               =     23.485464 s
Total packets            =          4461
Minimum delay            =      0.005605 s
Maximum delay            =      0.092024 s
Average delay            =      0.015609 s
Average jitter           =      0.001879 s
Delay standard deviation =      0.007593 s
Bytes received           =       2284032
Average bitrate          =    778.024058 Kbit/s
Average packet rate      =    189.947280 pkt/s
Packets dropped          =           236 (5.02 %)
Error lines              =             0
```

Vita

Name:	Jay Sethi
Place of Birth:	Winnipeg, Manitoba
Year of Birth:	1983
SecondaryEducation:	St. Paul's High School (1996 – 2001)
Honour and Awards:	Dean's Honour List (2002, 2003)
	Faculty of Engineering Second Year Scholarship
	Advanced Early Admission Scholarship

Name:	Matthew Wawrin
Place of Birth:	Winnipeg, Manitoba
Year of Birth:	1980
SecondaryEducation:	Grant Park High School (1992 – 1998)
Honour and Awards:	Dean's Honour List (2001, 2002, 2003)
	Advanced Early Admission Scholarship

Name:	Ryan Supeene
Place of Birth:	Winnipeg, Manitoba
Year of Birth:	1982
SecondaryEducation:	Glenlawn Collegiate (1997-2000)
Honour and Awards:	University Entrace Scholorship
	Linda Asper Scholorship
	Elected Senior Stick of the Faculty of Engineering for the 2005/2006 Year.
	Gold Metal Paper Technical Communications 2004